Algorithmes Branch&Bound Pair-à-Pair pour Grilles de Calcul

Mathieu Djamai

UNIVERSITÉ DE LILLE 1

ÉCOLE DOCTORALE SPI

Sciences Pour l'Ingénieur

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université Lille 1

Spécialité : INFORMATIQUE

Soutenue par

Mathieu DJAMAÏ

Algorithmes Branch-and-Bound Pair-à-Pair pour grilles de calcul

Directeurs de thèse: Pr. Nouredine MELAB
et Dr. Bilel DERBEL

préparée à INRIA Lille Nord-Europe, Équipe DOLPHIN

soutenue le 11 Mars 2013

Numéro d'ordre : 41079 — Année : 2013

Jury :

Rapporteurs :	Daniel TUYTTENS	-	Professeur, Université de Mons, Belgique
	Didier EL BAZ	-	Chargé de Recherche HDR, CNRS/LAAS
Examinateur :	Mohamed MOSBAH	-	Professeur, LaBRI, Université Bordeaux-1
Directeurs :	Nouredine MELAB	-	Professeur, Université Lille 1, Lille
	Bilel DERBEL	-	Maître de Conférences, Université Lille 1, Lille
Invité :	Mohand MEZMAZ	-	Université de Mons, Belgique

UNIVERSITY OF LILLE 1

DOCTORAL SCHOOL SPI

Engineering Sciences

PHD THESIS

to obtain the title of

Ph.D. of Science

of the University Lille 1

Specialty : Computer Science

Defended by

Mathieu Djamaï

Peer-to-Peer Branch-and-Bound Algorithms for Computational Grids

Thesis Advisors: Pr. Nouredine Melab
and Dr. Bilel Derbel

prepared at INRIA Lille Nord-Europe, Dolphin Team

defended on March 11^{th}, 2013

Order Number : 41079 — Year : 2013

Jury :

Reviewers :	Daniel Tuyttens	-	Professor, Université de Mons, Belgique
	Didier El Baz	-	Research Director, HDR, CNRS/LAAS
Examinator :	Mohamed Mosbah	-	Professeur, LaBRI, Université Bordeaux-1
Advisors :	Nouredine Melab	-	Professor, Université Lille 1, Lille
	Bilel Derbel	-	Assistant Professor, Université Lille 1, Lille
Invited :	Mohand Mezmaz	-	Research Scientist, Université de Mons, Belgique

Contents

List of Figures

Acknowledgments

I will seize the opportunity to thank warmly both my thesis advisors, Pr. Nouredine Melab and Dr. Bilel Derbel for their continuous support during the preparation of my thesis. I wish to thank Nouredine for being deeply involved in my work, for the numerous meetings with Bilel discussing my work, the articles and the presentations i have made during the thesis. I also thank especially Bilel for his great involvement during these three years, especially when writing and reviewing articles, reviewing and improving the manuscript, and at every step of the scientific work, when thinking about the challenges to face, desiging the solutions, proving the correctness of these solutions, defining the appropriate experiments.

I am also pleased to thank Pr. Daniel Tuyttens, Dr. Didier El Baz, Pr. Mohamed Mesbah as well as my advisors and Dr. Mohand Mezmaz for honouring me by their presence as examination jury for my thesis defense.

I also thank warmly my mother for supporting me during these three years (during my entire student life, in fact :)), my father for the many discussions we had on his experience of research in mathematics.

Last but not least, I shall not forget my dearest colleagues and friends (in and out of the Dolphin team) I have known during the preparation of my thesis, some of whom I had exchanged so many useful tips and valuable ideas : Mostepha Redouane Khouadjia, Ali Khanafer, ThÃ© Van Luong, Moustapha Diaby, Nadia Dahmani, Imen Chakroun, Marie-ÉlÃ©onore Marmion, Trong Tuan Vu, Pamela Wattebled, Russel Nzekwa, Jean Decoster, Julie Hamon, Yacine Kessaci, AhcÃ¨ne Bendjoudi, Khedidja Seridi, Martin Bue, Karima Boufaras, Ali Asim.

Chapter 1

Introduction

The PhD Thesis, presented in this document, deals with Large-Scale Combinatorial Optimization on Computational Grids. It has been completed within the DOLPHIN [1] research group from CNRS/LIFL , Inria Lille-Nord Europe and Université Lille 1. The presented works are part of the Inria HEMERA large-scale initiative, which involves several research teams using the Grid'5000 French experimental grid infrastructure.

Combinatorial Opimization Problems (COPs) are often NP-hard. Depending on the expected quality of the solutions, resolution methods for these problems fall into two categories: *near-optimal* methods, also referred to as heuristics, and *exact methods*. There also exists hybrid methods which combine complementary elements from multiple methods. Among heuristics, one can find *metaheuristics* which can be applied to a greater variety of problems. These methods are often used to tackle large problems and can produce good solutions within a short amount of time but these solutions are rarely optimal. On the contrary, exact methods allow to find optimal solutions along with the proof of optimality. However, their cost in terms of computation time is huge which makes them unpractical.

Large-scale parallelism, based on computational grids, appears to be a useful tool to face the processing cost issue of exact optimization methods. A grid can be seen as a set of resources located on multiple geographical sites, connected through a large-scale network and characterized by their volatility and heterogeneousness. Exploiting the potential computing power of such an environment requires the design of parallel algorithms which deal with the issues of volatility (for Fault-Tolerance purposes), heretogeneity (for Load Balancing purposes) and efficient communication management (for Scalability purposes).

During a resolution process, the irregularity of the search tree explored by exact methods as well as the heterogeneity of computational grids and their volatility, induce a huge amount of load balancing and checkpointing operations [Mezmaz 2007a]. All these operations imply high amounts of communications for storing and transferring work units between computing entities. Besides,

[1] Discrete multi-objective Optimization for Large-scale Problems with Hybrid dIstributed techNiques

the cost of such communications may be even much more important due to the scale of a grid and its communication delays. It becomes crucial to design appropriate mechanisms to deal efficiently with these issues. Most of existing works propose approaches based on the Master-Slave paradigm [Mezmaz 2007b, Drummond 2006, Mans 1995, V.K. Janakiram 1988], where, however, the roles of the master and the slave entities can vary greatly from one approach to another. In a general way, a slave entity is in charge of applying a given optimization method on a subproblem for the problem being solved. It computes a partial result which is sent to the master entity. The latter manages global information and is in charge of balancing workload among slave entities as well as detecting the termination of the computation (the moment when all the solutions have been explored).

However, every approach based on the Master-Slave paradigm faces a major limitation, related to scalability. Indeed, harnessing a huge amount of computational power to perform a task requires an intensive process of synchronization between the computing entities. In the case of an exact method, it consists in sharing the best solution found among all entities, balance the load over the network, handle checkpointing operations for fault-tolerance purposes. These tasks generally induce a communication bottleneck on the master entity, degrading significantly the performance of the approach.

To overcome this limitation, two classes of approaches have been explored in the literature. The first one consists in introducing additional levels of centralization. Such algorithms are better known as *hierarchical* Master-Slave algorithms [Bendjoudi 2009]. Additional "*sub-master*" processes are in charge of overseeing only a part of the whole network, including communication operations. These processes send synthesized information to the global master. This kind of approach reduces the communication load upon the global Master but introduces delays in processing slaves' requests as communications have to cross multiple levels of hierarchy.

The second class includes Peer-to-Peer (P2P) approaches. The Peer-to-Peer paradigm is usually used for data sharing. In this thesis, we believe that it can be used for computation purposes, as in [Nguyen 2012].In [Di Constanzo 2007], the main idea is to provide a generic platform, so that communications induced by a parallel algorithm can be handled into a P2P fashion, using routing mechanisms. In their experiments, they validate the platform by using a Master-Slave based Branch-and-Bound algorithm, named *Grid'BnB*. Computing entities act as peers, each one having a set of neighbors. At the application level, one of these peers acts as the master entity and other peers act as slaves. Thus, whenever a slave peer needs to communicate with the master, its message is relayed through multiple peers before being delivered to the master. The main drawback of such an approach is that the communication load on the master entity is momentarily reduced,

but it is distributed through time. Indeed, for a given number of slaves, this architecture simply allows one to delay the requests from the slaves located far away from the master in the Peer-to-Peer overlay network so that the master can process incoming requests within a more reasonable amount of time. The main issue inherent to a Master-Slave architecture, that is the communication bottleneck preventing the approach from being scalable, is not overcome. Another Peer-to-Peer approach has been designed by Bendjoudi *et al.* [Bendjoudi 2009]. The application is based on a Master-Slave architecture but direct communications between the computational entities are allowed for sharing the best solution and other collaboration tasks to reduce the load on the master. The main limitation is that this kind of optimization can be performed only in specific situations by defining a sort of P2P communication layer beneath the application level. The main operation of the algorithm still relies heavily on the master entity and thus, remains an obstacle to scalability. Moreover, to the best of our knowledge, none of these works provide a formal proof of the correctness of the designed approaches.

In this thesis, we propose a new approach to overcome this limitation of scalability. The approach is completely decentralized, that is computational entities operate in a fully Peer-to-Peer fashion. Designing adequate mechanisms under such a decentralized architecture is very challenging. Indeed, there is no entity in the network which has a global view of the network. In the case of the Branch-and-Bound algorithm, no entity can determine immediately what the best solution found so far is nor if the termination of the calculation has occurred. Whereas those two tasks can be handled easily in a centralized[2] environment, they become major challenges in a fully decentralized one. Thus, to face these challenges, our approach provides the following mechanisms. Each peer is in charge of handling a local work pool and sharing it with other peers. Global information, like the best solution found so far by the optimization method, is broadcast over the network by the peers. Termination detection is handled in an innovative and decentralized way. Each peer can detect locally the presence or absence of a work unit somewhere in the network only by communicating with its neighbors and using some of the network overlay's properties. Performing all the required synchronization operations in a fully distributed manner allows to harness resources at very high scales by reducing significantly the communication load upon the computational entities. In addition, we propose a formal proof of the correctness of our approach, that is, the termination of the computation is detected in an appropriate way, the exploration process is achieved in a finite amount of time and no deadlock situations can occur during communication operations.

Moreover, we provide an extensive experimental study on the influence of the network overlay's topology on the scalability of our approach. Indeed, to the best of our knowledge, almost none of the existing works take into account the

[2]Whether Master-Slave or hierarchical architectures.

way in which computing entities communicate with each other. Most works using decentralized architectures simply conduct experiments using a predefined P2P overlay and present the subsequent results as a proof of concept of the involved distributed algorithm. We consider various topologies among the most commonly used in the literature. Experiments show that our approach can operate independently from the used topology and that best results are achieved using small-world graphs, which offer the best compromise for distributing communications over the network.

Finally, we extend our approach to *dynamic* environments, that is where resources are volatile. We design additional mechanisms for checkpointing operations, reassigning work units from failed peers, and handling efficiently the joining or the departure of peers from the network. Experiments demonstrate the robustness of the approach when facing real-case as well as "*failure-intensive*" scenarios.

This thesis is organized into 7 chapters. Chapter 2 introduces some key concepts dealing with Combinatorial Optimization as well as Grid Computing. It also provides an overview of existing works among Master-Slave based approaches and P2P-based approaches. Chapter **??** describes our fully Peer-to-Peer approach into a *static* environment, that is where all the resources are considered to be reliable, as well as a complexity study in terms of time and messages. Chapter **??** details the formal proof of the correctness of our approach. Chapter **??** provides an extensive experimental study of our approach and the impact of the network topology on its performances. Chapter **??** describes our Fault-Tolerant approach and the different mechanisms designed to handle resources volatility. We also demonstrate the robustness of our approach by simulating various failure scenarios. In Chapter **??**, we summarize the major achievements of our contributions in this thesis and give some perspectives. Appendix A provides an additional study of Peer-to-Peer protocols.

Chapter 2

Parallel Distributed Branch-and-Bound Algorithms

Contents

2.1 Introduction

Many real-world problems can be modeled as Combinatorial Optimization Problems (COPs). Solving such problems in an exact manner may be computing-intensive and time-consuming and thus requires a huge amount of computational resources to be achieved. More and more computational resources can be made available from computational grids, clusters, personal computers connected through the internet, etc. Solving difficult and large scale combinatorial problems on these environments requires the design of efficient and scalable distributed algorithms that can be effectively deployed over large scale distributed environments. In this chapter, we introduce some key concepts at the crossroads of Combinatorial Optimization and Distributed Computing related to the issues underlying the design and the deployment of such algorithms. More precisely, Section 2.2 introduces the main concepts related to Combinatorial Optimization and particularly, the Branch-and-Bound algorithm. Section 2.3 introduces some concepts related to Grid Computing and presents an overview of Grid environments and some state-of-the-art works related to the parallelization of the B&B algorithm. Section 2.4 introduces more technical elements to be used in our contributions. More precisely, we introduce some elements from an existing approach and problem-encoding procedure which our contributions are based on.

2.2 Combinatorial Optimization and the Branch-and-Bound Algorithm

2.2.1 Preliminaries

Generally speaking, in *Combinatorial Optimization*, also referred to as *discrete optimization*, the goal is to find one or several configuration(s), among a finite but large set of possible configurations, optimizing a given objective function. A configuration is termed as a *solution* of the problem being considered and belongs to a mathematical space called *solution space*. Selecting a solution depends on its *cost*, which is defined by an *objective function*. Solving a combinatorial optimization problem then turns out to be the process of finding a solution with an 'accurate' value of its objective function. Depending of the expected accuracy, resolution methods can be classified into two categories (See Figure 2.1):

- **Exact methods:** Exact methods allow to obtain a set of solutions which are optimal with respect to the objective function. In other words, the output solutions are *guaranteed* to maximize (or minimize) the objective function. In this class of methods, one can cite Branch-and-Bound algorithms, constraint programming, dynamic programming, etc. Although an exact method allows to solve a given problem to optimality, it however requires to enumerate (in an implicit or explicit way) all the possible solutions for the problem being solved.

- **Approximate methods:** Approximate algorithms, also referred to as *heuristics*, allow one to obtain "good" solutions without strong guarantees on the optimality of computed solutions. They can be applied to problems for which the size of the solution space is subsequent. Some of these heuristics can be designed to solve a specific type of problems while others are more generic. The latter are called *metaheuristics*. These methods can operate in two different ways. They can be *single-solution based*, meaning that one solution is initially considered and then improved iteratively along with the solution space exploration process (Tabu search, simulated annealing, steep descent methods, hill climbing, etc). The other type is called *population-based*, meaning that an initial set of solutions is considered and then, all these solutions are improved simultaneously or independently, during the exploration process (Particle Swarm optimization, Ant Colonies, Evolutionary Algorithms, etc).

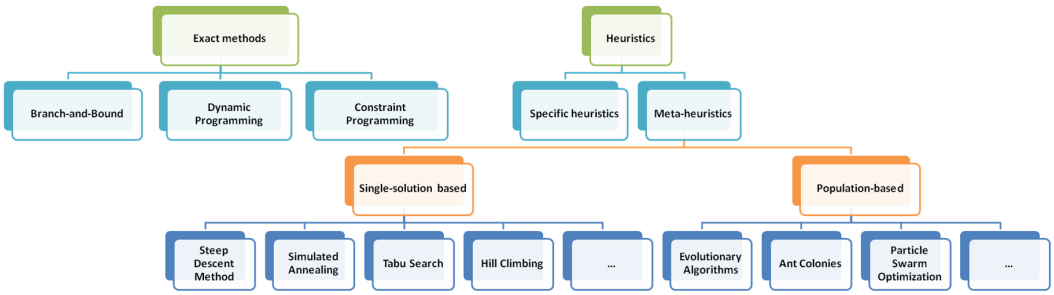

Figure 2.1: Taxonomy of resolution methods in combinatorial optimization.

The goal of this thesis is to design efficient *exact* algorithms for combinatorial optimization problems. In fact, although exact methods allow one to find a solution with optimality guarantees, they are computing intensive and very time consuming when tackling hard and large scale problem instances. In this context, parallel computing is more than just a possible alternative, it is in fact crucial to not say mandatory to solve such instances. The contribution of this thesis lies precisely in the design of parallel Branch-and-Bound algorithm in large scale distributed environments. In the next section, we give a brief overview of how a B&B algorithm operates before switching to a literature review on parallel approaches for B&B.

2.2.2 The Branch-and-Bound Algorithm

2.2.2.1 Main Operation

Let us consider a set S of all candidate solutions with respect to an optimization problem and a real-valued objective function f which associates a numerical value to every solution in S. Remark that a candidate solution is not necessary a feasible solution with respect to problem constraints, but all possible feasible solutions are assumed to be included in the set S. Without loss of generality, our goal is to find the solution(s) in S minimizing the value of the f function: $\arg\min_f(S) =$

$\left\{ s \in S \mid f(s) = \min_{r \in S} f(r) \right\}$. In the following, we assume that the set S of feasible solutions is finite, thus making our problem be a discrete optimization problem. The Branch-and-Bound algorithm [Gendron 1994, Papadimitriou 1998, Ralphs 2003] can be viewed as an implicit enumeration of all the possible solutions in S in order to find the best one(s) w.r.t function f. This algorithm takes its origins in the years 1960's: Land and Doig [Land 1960] designed an iterative algorithm to solve what they refer to as "Discrete Programming Problems". Dakin [Dakin 1965] later proposed improvements to this algorithm and proposed a tree-based algorithm for mixed integer programming. Those two algorithms were among the first one to be classified as "Branch-and-Bound" techniques, as defined by Little, Murty, Sweeney and Karel [Little 1963].

From a more technical point of view, this algorithm decomposes the original optimization problem into several subproblems of smaller size. The resulting search tree is explored by building a tree where the root node represents the entire problem to solve, inner nodes represent subproblems and leaves represent solutions for the problem being solved. This exploration process can be decomposed into four mechanisms: branching, bounding, elimination and selection.

1. First, the algorithm performs the branching operation. It is also referred to as decomposition and consists in partitioning the set of feasible solutions for a given problem into smaller subsets (subproblems) on which the same optimization problem applies. These subproblems are recursively decomposed until the solution level is reached (leaves in the search tree), or these subproblems are not likely to improve the best solution found so far.

2. Second, the algorithm bounds the generated subproblems. It computes a lower bound of the optimal solution for the problem, using data available from the generated subproblem. Different bounding functions are available in the literature. We use the well-known lower bound proposed by Lageweg et al. [Lageweg 1978] for the Flow-Shop Scheduling problem, based on the Johnson's algorithm [Johnson 1954].

3. Third, the B&B proceeds with an elimination process. The purpose is to avoid exploring subproblems that will likely not lead to the discovery of the optimal solution. This process is achieved in two steps. The algorithm (i) determines whether the solutions are feasible or not (and then discards non-feasible solutions) and (ii) compares for the remaining subproblems their lower bound with the best solution found so far[1] and discards subproblems whose bound is greater (for minimization problems).

4. The last step of the B&B exploration process is to determine a policy for exploring the remaining subproblems to be explored. More precisely, one has to determine which problem will be selected for exploration. Multiple

[1] Solution which can be seen as an upper bound of the optimal solution.

strategies exist in the literature and are described in detail in the following section.

Therefore, the Branch-and-Bound algorithm consists in recursively branching and bounding (sub)problems aside eliminating some of these and exploring the remaining ones according to a predefined strategy.

The previous exploration steps are repeated until all solutions are explicitly or implicitly visited. The B&B is then guaranteed to output the optimal feasible solution(s) for the problem being solved. It should be clear from the previous discussion that the B&B exploration process can be seen as a tree-based process. In fact, the original problem (whole set S) can be seen as the root of the tree and the subproblems resulting from its decomposition as its direct children nodes. The subproblems resulting from the decomposition of a subproblem can be recursively considered as children nodes, and so on until we end up with single solutions being the leaves of the tree.

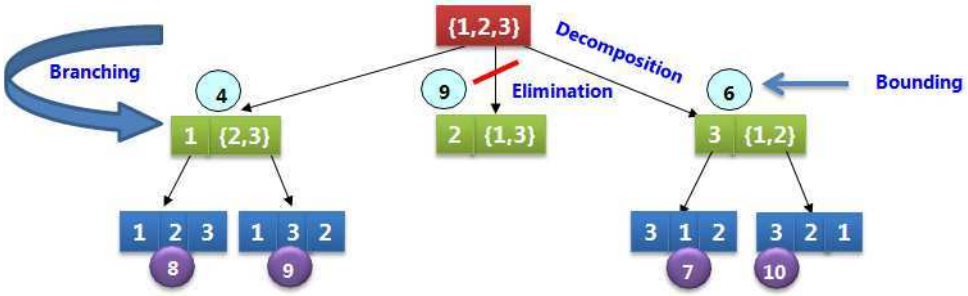

Figure 2.2: General illustration of the operation of the Branch-and-Bound algorithm for a 3-element permutation-based problem.

2.2.2.2 Exploration Strategies

After a problem is decomposed into subproblems, it is necessary to determine which subproblem will be explored next, according to a specified strategy. Three main exploration strategies can be distinguished:

- **Depth-First Strategy:** This policy assumes the existence of an arbitrary order among subproblems. When a problem is decomposed into several subproblems, this strategy consists in always exploring those subproblems according to the so-defined order. Let us consider a set of n subproblems ordered from S_1 to S_n at some iteration of the B&B algorithm. This strategy then begins exploring subproblem S_1 first. If this subproblem can be decomposed further, then the first (with respect to the considered order) of its subproblems will be explored first, then the second subproblem and so on recursively until the subtree rooted at S_1 is completely explored. When the exploration is done with subproblem S_1, then and only then the second subproblem S_2 can be

explored. The procedure is then recursively applied until the whole tree is explored. Note that this exploration process is independent from the numbering of the subproblems. Here, problems are labeled according to the order they were generated by the problem decomposition mechanism, that is S_1 is the first subproblem generated during the decomposition.

- **Best-First Strategy:** This strategy relies on the lower bounds obtained when evaluating generated subproblems. It assumes that, when a set of subproblems is generated, the optimal solution has a greater probability to be contained in the subproblem having the "*best*" (lowest) lower bound. When this subproblem is explored, the algorithm explores the subproblem whose lower bound is immediately higher than the previous one, and so on. This policy implicitly assumes that the bounding function is *discriminating*, meaning that, all the subproblems resulting from the same decomposition have different lower bound values. Nevertheless, this limitation can be overcome by defining a *sub-policy* to deal with *equal priority* subproblems.

- **Breadth-First Strategy:** This policy does not introduce any arbitrary order in the exploration of subproblems. When a problem is decomposed into several subproblems, all the generated subproblems are first evaluated then some may be discarded by the B&B algorithm. The others are again decomposed into new subproblems. More technically speaking, let us consider a problem B. The strategy consists in generating its $B_{i\in[1,n]}, n \in \mathbb{N}$ subproblems at once, evaluating them all, possibly branching them and then, for all the remaining B_i subproblems, their subproblems $B_{i,j\in[1,n']}$ are generated and so on. The main advantage of this strategy is that it generates many large subproblems at the beginning of its execution. In the case of the Branch-and-Bound algorithm, it enables the branching operator to discard potentially large sets of solutions and speed up the resolution process. One can note that this exploration strategy requires to store a potentially sizeable list of subproblems in memory.

Now that we reviewed the main generic components of a Branch-and-Bound algorithm, we give an overview on which parts of the algorithm can be parallelized and how this parallel strategies can be effectively deployed. This is the goal of the next Section where we start providing a general overview of grid-based environments.

2.3 Parallel Branch-and-bound literature overview

In this section, we give an overview of the different works existing in the literature as well as the parallel models of Branch-and-Bound algorithms. Deploying parallel B&B algorithms requires to take into account the characteristics of the target execution environment. Therefore, we shall first study these parallel environments as well as their characteristics.

2.3.1 Large scale Grid Environments

A *Grid* can be viewed as a set of computational resources scattered over several geographically distributed sites and connected through a wide-area network. In the following, we present the main properties defining more precisely a grid environment [Melab 2005].

2.3.1.1 Multiple administrative domains

First, the computing resources in a grid environment are typically distributed among multiple administrative domains and managed by different organizations. With respect to this property, one major issue appearing in grid computing is security. In fact, very often, users and resources providers can be clearly identified, which allows one to improve security. Nevertheless, communicating between multiple sites typically through *firewalls* can be an obstacle and pauses challenging issues. In global computational systems like XtremWeb [Fedak 2003], which are based on Internet-wide cycle stealing, this issue can be overcome quite easily as these systems are based on voluntary computing, meaning that computational units initiate communications from inside the administrative domain. In Condor [Litzkow 1988], multiple approaches for harnessing multi-domain resources as if they all belong to a unique domain, have been designed such as *Condor-G* and *Condor Glide-in* [Frey 2002] or *flocking*. The two first approaches are coupled with the Globus platform [Foster 1997] whereas the latter solution consists in implementing a resource manager for each administrative domain. These resources managers exchange the tasks which can not be processed locally.

2.3.1.2 Heterogeneity

A grid is by essence heterogenous. Resources heterogeneity, whether at the hardware level or the software level, can be viewed as the consequence of the large variety of equipments composing the grid and the large variety of software tools that could be run on them. A grid can integrate hardware from multiple vendors, run various operating systems, and use different network protocols for remote communication, etc. In contrast, a single site of the grid is often composed of homogeneous resources, mostly aggregated into clusters.

2.3.1.3 Large-Scale systems

Due to the number of available computational units and the wide-area network interconnection infrastructure, a computational grid is a large-scale system. Depending on the considered scale, a distinction is generally made between computational grids on the one hand, and global/P2P computing systems on the other hand. Those latter are usually believed to enable the aggregation of much more computational resources and are dedicated to voluntary computing. One of the most famous voluntary computing systems is *SETI@Home* [Milojicic 2008, Anderson 2002]. We can

also cite XtremWeb [Fedak 2003] which can be considered as a voluntary computing system but serves more general purposes as it aims to convert any set of resources into a fully operational grid-like environment. In such large scale systems, computing intensive applications are not guaranteed to scale with the system. This is typically due to communication latency and load imbalance issues occurring in such heterogenous and large scale networked systems. Designing distributed protocols which are both scalable and efficient is one of the most challenging tasks to achieve high performance using grid environments.

2.3.1.4 Dynamic environment

The last major property characterizing a grid environment is resources' volatility. By volatility, we mean the fact that computing resources are not expected to be always available for the application. This is due to hardware crash, software issues or any other system variance. Volatility should not be viewed as unlikely or as an exception in a grid environment. In fact, the probability that resources are operational and available for the application is usually observed to decrease as the amount of aggregated resources increases. Volatility poses several challenging issues such as: dynamic resource discovery, fault tolerance, data recovery, synchronization, etc. These issues are difficult to deal with at a hardware level as they are mainly dependent on the nature of the application being executed. They are instead tackled mostly at the application level.

2.3.2 Parallel models for the B&B

The Branch-and-Bound algorithm can be parallelized according to multiple models, each one focusing on a specific element of the algorithm. Here, we give an overview of these different models based on the classification proposed by [Gendron 1994, Cung 1994, Melab 2005]. Four models have been identified: *multi-parametric parallel model, tree-based exploration parallel model, parallel bounds evaluation model* and *the parallel evaluation of one bound*.

2.3.2.1 Multi-parametric parallel model

The multi-parametric parallel model, not often studied in the literature, consists in considering multiple B&B algorithms (See Figure 2.3), which is a coarse-grained model. Multiple variants of this model can be derived by modifying one or several parameter(s) of the algorithms. These algorithms may only differ through the separation operator (used to split a problem into sub-problems) in [Miller 1993], the selection operator (determining the order in which sub-problems are explored) in [V.K. Janakiram 1988] where a variant of the *depth-first* strategy is used for exploration. Each algorithm randomly selects the next subproblem to explore among the last generated subproblems. In [Kumar 1984], each algorithm uses a different upper bound in the experiments. The main idea is that only one algorithm uses the best solution found so far while others use this value increased by a value $\varepsilon > 0$. Another

version of this parallel model consists in splitting the interval composed of the lower bound for the problem and the best solution found so far into sub-intervals. Each sub-interval is assigned to one of the algorithms.

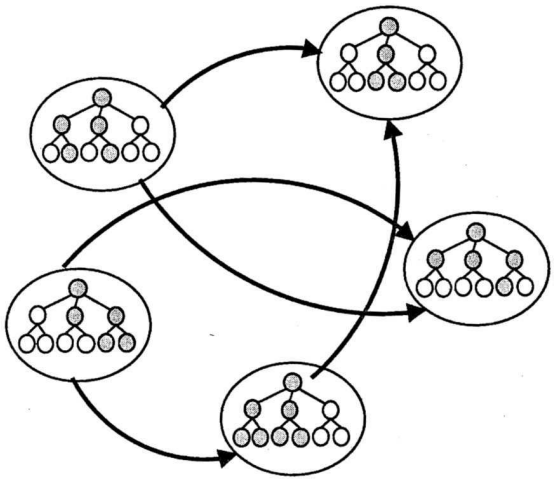

Figure 2.3: Illustration of the Multi-Parametric parallel model.

The main advantage of the multi-parametric parallel model is its genericity, enabling its use transparently for the user. Its main drawback is the additional cost of exploration as some nodes of the exploration tree are visited multiple times. Besides, as the number of algorithms involved is low, this model shall be used on grids along with other parallel models.

2.3.2.2 Tree-based exploration parallel model

The tree-based exploration parallel model consists in exploring in parallel multiple subtrees corresponding to subsets of the search space for the problem. (See Figure 2.4). This implies that the separation, selection, evaluation and branching operations are executed in parallel, (a) synchronously by different algorithms exploring these sub-spaces. In the synchronous mode, the Branch-and-Bound algorithm contains multiple phases. During each phase, algorithms perform exploration independently from each other. Between the phases, algorithms synchronize to exchange information, like the best solution found so far. In the asynchronous mode, algorithms communicate unpredictably.

In comparison with other models, the tree-based exploration parallel model is more attractive and is the main focus in many works for two main reasons. On the one hand, the degree of parallelism of this model can be very high for large-scale problems, which could justify by itself the use of a computational grid. On the other hand, the implementation may be faced to multiple parallel programming challenges. Among these issues, one can cite the localization and the management of the list of sub-problems to solve, the load balancing, the communication of the best solution found, the detection of the algorithm's termination, and the fault tolerance.

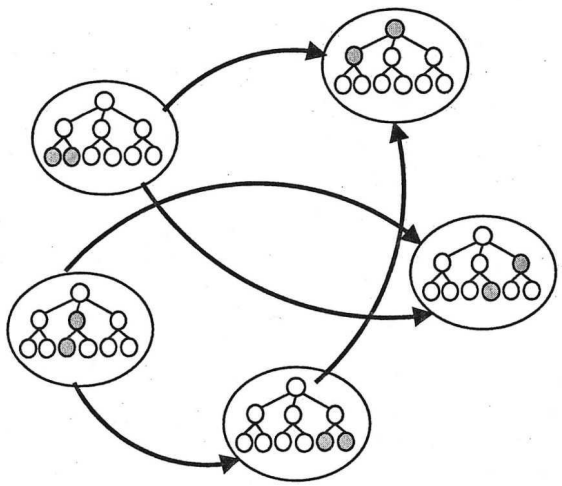

Figure 2.4: Illustration of the Tree-Based Exploration parallel model.

In [Gendron 1994], one can find an analysis of synchronous and asynchronous Branch-and-Bound algorithms. It appears that synchronization comes to be useless and inefficient on homogeneous machines with message passing features, consisting of less than 32 processors. Its usage on computational grids (heterogeneous and volatile) is not recommended. Still in this article, a study has also been conducted on the problem of localizing and managing the remaining sub-problems. Two categories of algorithms have been identified: those using a unique list of subproblems, shared by all processes and those using multiple lists. In the conclusions of this study, the authors claim that the use of a single list can be interesting for problems where the calculation of the bound is not trivial, and for machines (especially with shared memory) with a low number of processors. Therefore, the use of a single list is not recommended when using grids. In our works, we will focus only on asynchronous algorithms using multiple lists to store generated subproblems as they can be seen as more adapted for grids. Three approaches exist for managing these lists: *collegial, grouped* and *mixed* [Gendron 1994]. In the collegial approach, each process manages its own list where it stores the subproblems it generates. The grouped organization consists in defining sets of processes. A global list is defined for these processes to store all the subproblems they generate. The mixed approach is a collegial approach where each process manages its own list and simultaneously shares a global list with all other processes. As the collegial approach is fully distributed, it can be expected to have a greater scalability. However, the mixed approach can be efficient for grids with multiple administrative domains: processes belonging to the same domain can share a common list.

The load balancing issue consists in minimizing the number of situations where processes have many sub-problems to explore while others have empty lists. A load balancing policy is necessary. Its aim is to initiate, at some appropriate times, transfers of sub-problems from overloaded machines to under-used machines. Three agents define a load balancing policy: an information agent, a transfer agent and

a localization agent [Melab 1996, Melab 1997]. The first agent collects information about the load of processors. It provides a load indicator (number of sub-problems, CPU usage, ...), which allows to define the load of a machine, and an information exchange strategy (centralized, distributed or hierarchical). The transfer agent can use a passive, active or mixed strategy. In the passive strategy (respectively active), transfers are initiated by under-used (resp. overloaded) processors. The mixed strategy combines both. The localization agent allows to determine the receiving process (often the least used) for the sub-problems in excess in the overloaded machines. In a grid-like environment, to take into account the machines' heterogeneity, their power should be included into the definition of the load indicator. Gathering information at large scales can induce a communication bottleneck when the global load increases rapidly. In the large-scale cycle-stealing model, used in our works, heterogeneity is considered in a more natural way: powerful machines ask for more work to weaker machines.

The best solution found by a process can be communicated to other processes whether through a shared storage space or by broadcasting mechanisms. The first approach consists in the use of a global variable to store the best solution found so far. This variable is updated each time a process finds a solution better than the current one. The communication of this value to other processes can be achieved through broadcasting or using any approach based on propagation through neighbors. The broadcasting method shall not be considered in large scale environments. The issue of termination detection is not specific to B&B algorithms. In our works, a process detects termination through gathering information from other processes.

The fault tolerance issue for exact exploration methods is crucial. Indeed, the loss of one or more sub-problems can prevent the processes from discovering the optimal solution for the problem being solved. On a volatile computational grid, the issue can be handled whether at the middleware level of at the application level. At the middleware level, the proposed solutions are often independent from applications. The failed process is restarted from scratch, which can be inefficient for processes with long lifetimes. At the application level, the issue is handled through a *checkpointing* mechanism. It allows to restart a failed process from its last saved state. Thus, it is crucial to determine which data must be saved. The data saved by each process usually contains the best solution found so far, the current problem being explored and the list of pending sub-problems. Other data can be saved if the current parallel model is combined with other models.

2.3.2.3 Parallel evaluation of solutions/bounds model

The Parallel evaluation of solutions/bounds Model can be compared to the Parallel Population Evaluation Model, often used with metaheuristics. This model is data parallel and allows the parallelization of the generated subproblems. It can be used when the evaluation of the bounds is performed entirely after the generation of the sub-problems. The B&B algorithm is not changed: only the evaluation phase becomes faster. The main advantage of this method is its genericity (See Figure 2.5).

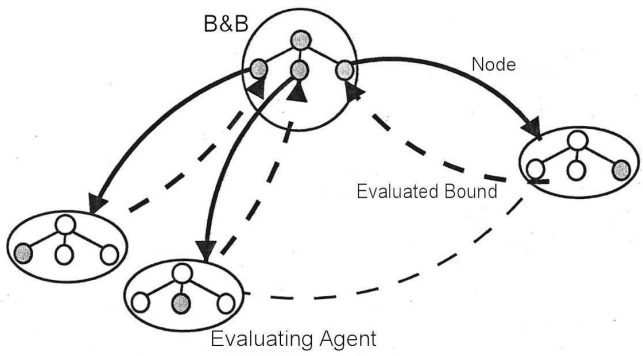

Figure 2.5: Illustration of the Parallel evaluation of solutions/bounds model.

However, in a grid environment, it can be inefficient for the following reasons: (i) the model is intrinsically asynchronous, which makes it costly in terms of CPU time within a volatile and heterogeneous environment; (ii) its granularity (the cost of the bound evaluation function) can be fine and thus, inefficient in large-scale environments. For instance, in the case of the Flow-Shop Problem, the processing cost of the bound's evaluation may not be high enough to justify its parallelization. (iii) The degree of parallelism of this model depends on the problem being solved. It is often limited and decreases along with the exploration process: the number of generated sub-problems decreases when the number of constraints for the problem splitting operation increases. Combining this model with the tree-based exploration model can induce a massive parallelism which can be efficiently employed only within grid environments.

2.3.2.4 The Parallel Evaluation model for a single bound/objective function

This model can be used for the resolution of real-world problems where the evaluation of the objective function/lower bound requires to access to massive amounts of data that can not be handled on a single machine. Because of this hardware constraint, those data are distributed among multiple sites. The evaluation of the objective function takes advantage of the data parallelism. The parallel evaluation of the objective function can be interesting when it is costly in terms of time.

This model requires the definition of new specific elements for the problem being processed, like partial objective functions and a function to aggregate these partial results. As the implementation of this model is naturally synchronous, it is crucial to memorize all the partial evaluation value for the solution being evaluated, to manage fault tolerance and variable availability of the resources in a grid. As its scalability can be very limited, this model shall be combined with other models.

2.3.3 Deploying parallel B&B

Having these different parallel strategies in mind, two main paradigms can used to make a parallel B&B algorithm effective and to deploy it over a distributed computing environment, namely, the Master-Slave model and the P2P model. In the next sections, we give a literature review on these models while focusing on their main characteristics with respect to B&B.

2.3.3.1 Master-slave approaches

In [Aida 2002, Aida 2005], a Master/Slave-based parallel B&B algorithm is proposed and deployed on a grid. This approach is aimed to avoid performance degradation caused by the communication overhead between the master process and worker processes. Processes are organized into sets, where each set comprises a group of worker processes and one master process to coordinate them. In addition, a process called *Supervisor* is in charge of controlling and coordinating all the sets of processes. One set of worker processes explores a given part of the search tree. The supervisor assigns a subset of solutions to the master of the set and this master dispatches the work to its worker processes. One can see the supervisor as an entity performing load balancing operations between the sets of processes. This supervisor, as well as the master process of each set of processes, is in charge of gathering and broadcasting the best solution found so far, thus accelerating the exploration process. The latter approach shows a limited scalability as it may create a bottleneck on the Master processes and the Supervisor process. The authors discuss the granularity of tasks, notably when tasks are fine-grained, the communication overhead is too high compared to the computation of tasks. The algorithm has been implemented using GridRPC middleware [Seymour 2002], Ninf-G [Tanaka 2003], and Ninf [Sato 1997].

The granularity issue is studied in [Di Constanzo 2007] but yet a hierarchical Master/Slave model is used therein. The architecture of the approach, named *Grid'BnB* is quite similar whereas the communication layer is a bit different. The application's design is to fit a grid environment. It is composed of four different types of entities: *master*, *sub-master*, *worker*, and *leader*. The master has the same role as the supervisor in the previous approach. The sub-master is in charge of coordinating one set of worker processes. The difference comes from the *leader* role. The approach assumes that the physical architecture is cluster-based. Each set of processes comprises several workers and is deployed on a physical cluster. The cluster running the master process also hosts the sub-master processes which are in charge of communicating with other clusters (see Figure 2.6). In those other clusters, one worker is chosen to be the *leader*. It is given a specific role, which is to handle communications with its sub-master. Thus, when a worker discovers a new best solution for the problem being solved, it broadcasts it to all the workers belonging to the same cluster, including the leader. This leader sends it to its sub-master process, which broadcasts it to the whole network through the master

process.

A middleware has been designed from this application. It enables to emulate Master-Slave based parallel B&B techniques on P2P networks or platforms. The main difference with a classical hierarchical Master-Slave model is that messages are relayed through peers towards the master and the sub-masters. This approach is less exposed to a communication bottleneck issue but it may end up with huge communication delays as these communications can be relayed through a great number of entities before reaching their destination point. The main goal of [Di Constanzo 2007] is in fact to design a middleware that hides the network architecture/topology for computational applications.

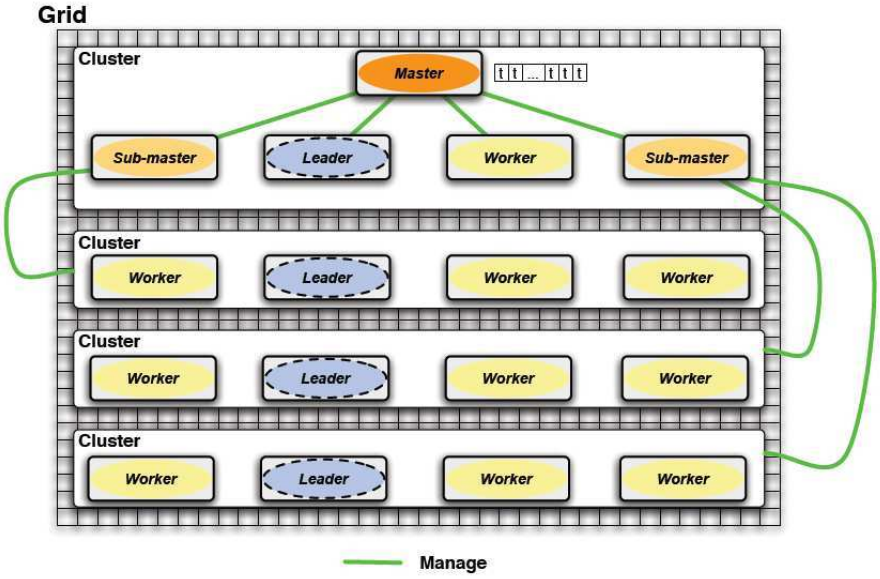

Figure 2.6: The Grid'BnB architecture

Xu *et al.* [Y. Xu 2005] and Eckstein *et al.* [Eckstein 2000] proposed respectively *ALPS* and *PICO* which are parallel B&Bs based on Master-Hub-Worker in which a layer of medium-level management is inserted between the master and the workers where each hub manages a static set of workers. They consider in their approaches cluster-based architecture. Each cluster contains a local Hub and one or multiple worker(s). The number of hubs increases with the number of workers and they avoid becoming overburdened by limiting the number of workers by cluster. Therefore, some computational burden is moved from the master to the hubs.

In [Drummond 2006], Drummond *et al.* propose another hierarchical B&B algorithm designed for grid environments. This approach is applied to the *Steiner* problem in graphs. In this problem , the branching operation creates two new sub-

problems. The approach performs the following operations: a master is run on the processor of one given cluster. This entity runs and monitors a set of *leader* processes, one on each cluster. Those *leaders* represent the processors of the cluster on which they are running. A leader splits its subproblem into two *left* and *right* subproblems. The *right* one is assigned to another leader. This process is repeated until no leader is available. Then, each leader processes its subproblem by sharing it with the workers being run on its own cluster.

In [Bendjoudi 2009], the authors suggest a hierarchical architecture to allow workers to communicate directly together after receiving a task from the master, the redundancy induced by [Mezmaz 2005]'s approach when exploring the search space can be significantly reduced.

Bendjoudi *et al.* [Bendjoudi 2011] have proposed a fault tolerant hierarchical B&B (See Figure 2.7), named FTH-B&B, in order to deal with the fault tolerance and scalability issues in large scale unreliable environments. Their algorithm is composed of several fault tolerant M/W-based B&Bs, organized hierarchically. Some other works, e.g., Cabani *et al.*'s PHAC ([Cabani 2007]), Saffre *et al.*'s Hypergrid [Saffre 2003], aim to dynamically redesign the topology in order to face communication bottleneck issues.

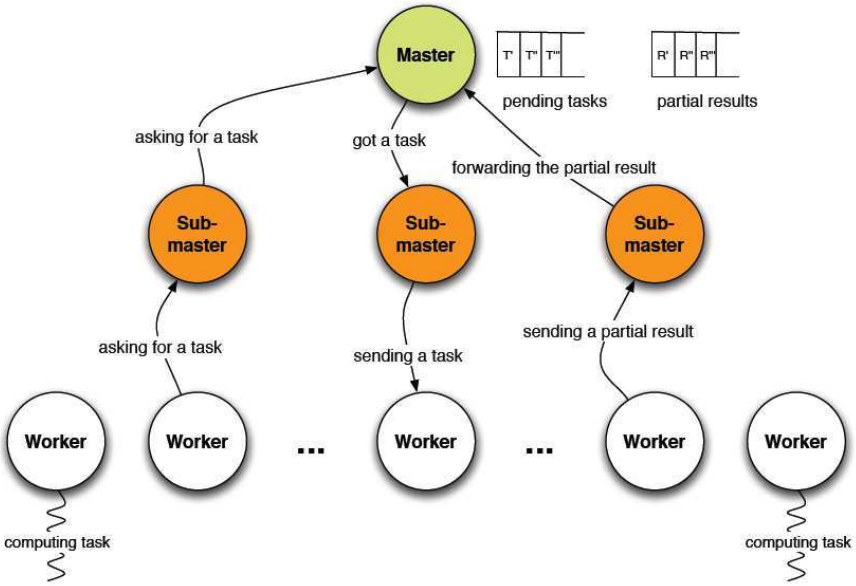

Figure 2.7: Illustration of a Hierarchical Master-Slave based architecture

In [Mezmaz 2005], an efficient encoding of the search space is proposed to reduce the size of exchanged messages. The overall parallel efficiency is then improved compared to previous solutions. The approach of [Mezmaz 2005] can be considered as the best parallel Master-Slave B&B approach that can be applied in a large scale computational environment such as grids [Mezmaz 2007b]. In particular, it was successfully applied to find the optimal solution of an unsolved Flow-shop hard instance, namely the Ta056 instance [Taillard 1993, Mezmaz 2007b]. The latter

approach is also based on the Master-Slave model. We describe it more in details later.

2.3.3.2 Peer-to-Peer approaches

Parallel applications designed under the Master-Slave paradigm may often face scalability issues due to bottlenecks on the master. To overcome this limitation, some works consider the use of the Peer-to-Peer (P2P) paradigm for parallel B&B as in *DIB* by Finkel *et al.* [Finkel 1987] and in the work proposed by Iamnitchi *et al.* in [Iamnitchi 2000]. The latter proposes a fully decentralized approach for the Branch-and-Bound algorithm. The role of each process is to manage a local work pool and share it with other processes whenever they receive a request. The best solution is broadcast over the network through the most frequently sent messages. By distributing the communication load among multiple processes, this approach gains in terms of scalability.

Instead of dealing with the scalability issues at the applicative level, Di Constanzo *et al.* [Di Constanzo 2007, Caromel 2007] propose a more generic approach operating at the communication layer. The approach consists in defining a fully P2P infrastructure and providing an information routing mechanism. Processes are organized in a P2P fashion: one of them acts as the master and all others as slaves. Whenever a slave needs to communicate with the master, its messages are routed/relayed by multiple peers before reaching the Master. While this approach enables to provide a better scalability, it only distributes the communication load of the master through time and introduces additional delays for processing slaves' requests.

More generally, P2P systems are used in wider fields of application, like voluntary computing as in SETI@HOME [Anderson 2002] where a central server collecting radio signals to be processed dispatches the tasks among personal machines provided by individuals around the world. Other systems were designed for content sharing purposes like Bittorent [Bittorrent 2005], Kademilia [Maymounkov 2002], Pastry [Rowstron 2001a], for massively distributed computing like Javelin [Cappello 1997], for data storage and retrieval like OceanStore [Kubiatowicz 2000], Freenet [Clarke 2001], NaradaBrokering [Fox 2005], PAST [Druschel 2001].

With respect to the large body of literature dedicated to Master-Slave based parallel B&B, there is relatively few works dealing with parallel B&B in P2P networks. In this thesis, we propose to take benefit from the scalability properties of P2P network in order to design new distributed B&B algorithms that are able to handle a huge amount of computational resources. More precisely, we leverage the tree-based Master slave approach of [Mezmaz 2005, Mezmaz 2007b] by providing a fully distributed alternative. Although, the approach described in this thesis and the one of [Mezmaz 2005, Mezmaz 2007b] share the same B&B work encoding, they are fundamentally different, since the underlying distributed protocols are different. In the next section, we give a more detailed overview of the parallel approach

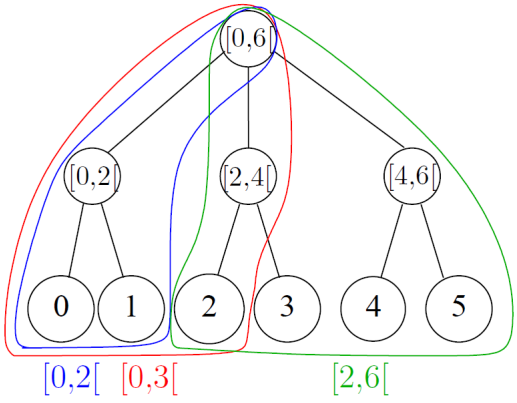

Figure 2.8: Example with a 3-variable permutation.

of [Mezmaz 2005, Mezmaz 2007b] which allows the reader to better appreciate our contributions.

2.4 The B&B@Grid approach

In this section, we analyze more thoroughly the Master-Slave based parallelization of the Branch-and-Bound Algorithm designed by Mezmaz *et al.* [Mezmaz 2005, Mezmaz 2007b], named B&B@Grid, which is, to the best of our knowledge, the most scalable 'Master-Slave'-based approach in the literature. This approach was designed for grid environments and parallelizes the B&B algorithm according to the tree-based exploration parallel model presented in Section 2.3.2.2. Some of the mechanisms of this approach are reused or referred to in our works.

2.4.1 Tree encoding

For the sake of clarity let us assume that we are given a permutation-like optimization problem, that is a problem where solutions can be encoded by mean of a permutation of size N. Hence, a basic B&B strategy can be represented by a tree where the root designates the problem to be solved, a leaf represents a solution (a permutation) and a node inside the tree represents a partial solution (equivalently, a sub-problem) where only some variables in the permutation are fixed. From a parallel/distributed point of view, many B&B algorithms are based on depth-first strategy when exploring the search space [Zhang 2000, Prieditis 1998, Mans 1995]. A sequential depth-first strategy explores the tree in a depth-first manner branching and bounding according to some branching and bounding policies. A parallel version of this strategy is to run several depth-first explorations in parallel on different parts of the search tree. As soon as a new best solution for the problem is found, it is communicated to other processes which allows them to update their own local best solution and thus to speed up the overall search process.

One major difficulty when setting up this general idea in a distributed environment is to carefully encode the nodes of the B&B tree in order to reduce the cost of exchanging messages between computational nodes and decide which part of the tree should be explored by which computational node. A trivial approach would be to encode a sub-problem of the B&B tree in a fully-comprehensive way by providing all necessary information. In [Iamnitchi 2000], a tree node (a subproblem) is encoded as a path from the root to the node itself which is still expensive. In [Mezmaz 2005, Mezmaz 2007b], an extremely simple and efficient encoding is proposed. Roughly speaking, the tree is labeled in such a way a sub-set of nodes (corresponding to a subproblem) can be encoded by an interval, i.e., two integers. A centralized model is then adopted to distribute intervals among computational nodes. More specifically, the approach described in [Mezmaz 2005, Mezmaz 2007b] consists in defining a central computational entity (the master) which is in charge of controlling intervals (work units) assignment to other computational nodes (the slaves).

In this thesis, we adopt the interval-based encoding of [Mezmaz 2005] while removing the need of the central coordinator. We briefly give an example to illustrate the tree encoding. For the sake of clarity and to make our results more comprehensive, we also recall the basic ideas of the Master-Slave approach and the main challenges we are addressing. Figure 2.8 gives a simple example with a p-variable permutation problem, (with $p = 3$) i.e., the optimal solution of our problem is one of the $3! = 6$ possible ordering of our variables. More precisely, a label (integer) is assigned to each leaf of the tree, corresponding to a specific permutation. Then, an interval $[x, y[\subseteq [0, p![$ refers to a subtree of the whole search tree. For instance, one can see in Figure 2.8 the subtrees corresponding to intervals $[0, 3[$, $[0, 2[$, and $[2, 6[$. Having this labeling, two operators, defined in [Mezmaz 2005], allow us to switch from the tree representation to the interval-based representation and conversely. Notice that exploring intervals $[0, 3[$ and $[2, 6[$ in parallel implies exploring the same region of the search space (permutation 3) twice, while exploring only interval $[0, 2[$ and $[2, 6[$ may fail to produce the optimal solution since permutation 2 is not explored.

2.4.2 Master-Slave tree exploration

In [Mezmaz 2007b], this tree encoding is coupled with a Master-Slave model. More precisely, the master owns initially the whole interval to be explored, i.e., the whole tree. Then, slaves ask the master for a sub-interval (a piece of the tree) to explore. The master continuously removes from the list the subintervals that are already explored and distributes those not explored yet. However, since many slaves having different computational power can ask for some work at the same time, some Slaves may end exploring the same part of the search tree inducing a so-called "*redundancy*". Redundancy issue is highlighted in [Mezmaz 2007b] and work dealing with this issue can be found in [Kumar 1984]. We also remark that each time a slave finds a new solution better than the best solution found so far, i.e., an ordering of

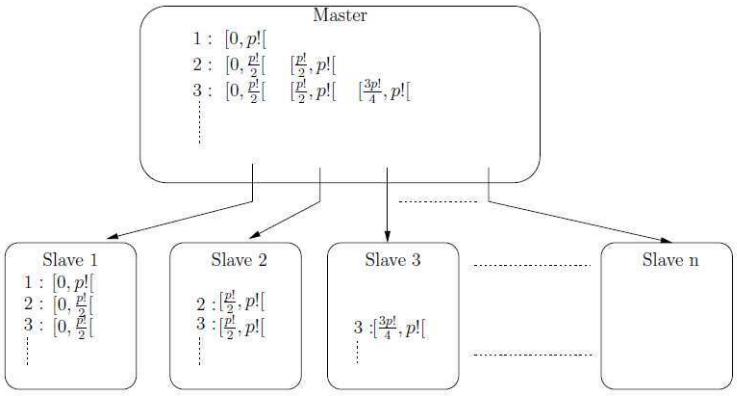

Figure 2.9: Operation of a Master-Slave based architecture.

variables, it directly informs the master which broadcasts this information to other slaves. Hence, sharing the best known solution is straightforward and does not induce any technical difficulty from a pure distributed/parallel point of view. In addition, termination of the whole distributed search process can be easily detected by the master, i.e. when the list of remaining sub-intervals maintained by the Master becomes empty. Load balancing is also quite simple since the master can control slaves progress and reassign work unit directly to under-used slaves.

A concrete scenario is proposed in Figure 2.9. Assume that we have initially n slaves. Assume that a first slave wants to start participating in the exploration process. Then, it sends a work request to the master. When receiving that request, the master answers with the whole problem to process (since until now the master has not received any request). In the second step, suppose that the master receives a work request from a second slave. To balance the load, the master splits the original interval into two parts and shares it equally between both slaves. Now, suppose that in the third step, a new request coming from slave 3 reaches the master. The master shall choose one of the already assigned intervals, split it into two parts and assign one of them to Slave 3. In the scenario of Figure 2.9, the Master chooses the interval assigned to slave 2 and shares it with slave 3. However, at this point of the execution, slave 2 is not aware of its newly assigned sub-interval $[\frac{p!}{2}, \frac{3p!}{4}[$. As depicted in Figure 2.9, it may explore the solutions belonging to interval $[\frac{3p!}{4}, p![$, i.e. the interval assigned to slave 3, leading to "*redundancy*". This issue is handled in [Mezmaz 2007b] as following. The master and the slaves continuously checkpoint together and update their intervals each fixed period of time. Depending on search speed and on whenever the checkpoint is done, redundancy may or may not occur. In our example, slave 1 has updated its work unit whereas slave 2 has not yet. However, in [Mezmaz 2007b] this solution was shown to be efficient. More precisely, experimental results have shown that the redundancy rate is of 0.39%. While this rate may appear to be low, as we are dealing with large-scale instances, it represents a huge amount of processing time. The associated total computation cost could be

highly significant if the exploration time of each node is high.

2.5 Conclusion

In this chapter, we provided a comprehensive description of the Branch-and-Bound algorithm and gave an overview of the main contributions related to the parallelization of this algorithm. We introduced the interval-based encoding of parallel B&B tasks which we use through this thesis. We also highlight the main characteristics of existing Master-Slave approaches for B&B. The rest of this thesis is devoted to provide a fully distributed P2P approach to Master-Slave approaches and assessing its efficiency.

Appendix A

Peer-to-Peer Protocols

Contents

This section deals with the main constraints related to the Peer-to-Peer technology, which can have a non-negiligible impact on the deployment and the performances of Peer-to-Peer applications and systems. In [Milojicic 2008], authors provide a detailed analysis of the main characteristics of any Peer-to-Peer networks.

A.1 Main characteristics of a P2P network.

A.1.1 Decentralization

Peer-to-Peer models requestion the concept of storing and processing data using centralized servers and accessing this data through "*Request-Reply*"-based protocols. In traditional *Client-Server* models, data is stored on a few main servers and is communicated, through network infrastructures, to client machines acting as user interfaces. Such centralized systems are best-fit for particular applications or tasks. For instance, security-based accesses can be better handled using such systems. However, this kind of architecture (Client-Server) can degrade performances because of communication bottlenecks or non-optimal resource usage. Although hardware performances and costs have improved, centralizing the sources of information can remain expensive in terms of configuration and maintenance. Human intervention maigh still be necessary to guarantee the coherence and non-obsoleteness of the content.

One of the key ideas behind decentralization is the influence that every user of the P2P network can have on data and resources. In a completely distributed system, every peer contributes equally to the network. Thus, implementing P2P can be very difficult to implement, technically speaking as there is no central entity having a global view of the network as well as the information stored in it. This is why many P2P systems are based on hybrid approaches like Napster [?], where a centralized resource directory is used and peers directly exchange data.

In fully decentralized systems, like Freenet [Clarke 2001] and Gnutella [Milojicic 2008], joining the network can be challenging. In Gnutella, for instance, new peers must be aware of a given number of pre-existing peers belonging to the network. They can also use a list of IP addresses of existing peers. The newcomer joins the network by connecting to at least one of them. From that point, it can proceed to discover other peers and gather their network information.

From a historical point of view, the first great project of Distributed Computing was the *Great Internet Mersenne Prime Search* (GIMPS, www.mersenne.org), whose objective was to harness computational resources distributed over the Internet network in order to calculate Mersenne's prime numbers. The GIMPS project initially used mails for communication purposes as this protocol is intrinsically decentralized. When a central entity assigns work units to unavailable computing resources, mail servers put corresponding mails into a waiting queue and send it again as soon as the resources becomes available again.

Then, another Distributed Computing project appeared (*"distributed.net"*), which was dedicated to decryption/decyphering. Initially, only one central server was used, which caused issues related to availability (Sometimes, for multiple weeks in-a-row). Therefore, a two-level proxy-system was used, on which a version of the server entity could be deployed and act as a proxy. Practically, task scheduling and assignment was easy as work coding was very simple : each work unit was referred to by a key and a set of work units could be represented by an interval of two integers.

P2P systems can be classified into two categories depending of their level of autonomy : "Pure P2P" and "Hybrid P2P". Of course, the classification can be made more precisely as shown in Figure A.1. This autonomy can have a direct impact of the self-organization ability and the scalability of a system, as the most decentralized systems are loosely linked to the network's or machines' infrastructure.

A.1.2 Scalability

One immediate advantage of decentralization is scalability. It can be limited by various parameters like the number of "*centralized*" operations (i.e. synchronization or coordination operations) which have to be performed, the application's intrinsic parallelism as well as the programming model used to design the calculation process.

Napster faced the scalability issue by making peers download music files directly from the peers having these files. Thus, Napster gathered more than six million users at its highest. On the opposite side, SETI@Home [2001] [?] focuses on a unique task that is inherently parallel. The objective is to harness computational resources

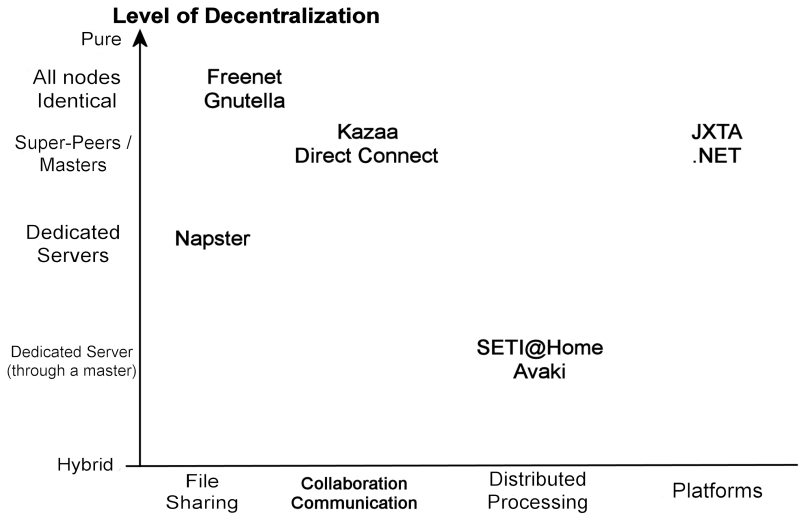

Figure A.1: Degrees of decentralization into various P2P networks.

over Internet to analyze data collected from telescopes in the hope of discovering extraterrestrial life forms. This platform accounts for almost 3.5 million users. Other systems like Avaki try to face the issue of scalability through a distributed object programming model.

Scalability also depends on the ratio between communications and computation between P2P nodes. Some computing-intensive application like decrypting or finding prime numbers, spend most of the time to computation, thus having a ratio close to zero. This makes these systems very extensible. In other situations, as in SETI@HOME, a communication bottleneck appears around the central entity during intensive data exchanges. For instance, one megabyte of data requires 10 hours of computation to be processed, necessitating a network bandwidth equal to 1 MB multiplied by the number of computational entities involved in its processing. There exists other applications, like, graphical rendering, that require thousands of machines to create an animation within hours instead of a year. However, this application induces a lot of data transfers, which makes it much less extensible.

A.1.3 Ad-hoc property

First data-sharing P2P systems like Gnutella [2001] et Freenet [Clarke 2001] are ad-hoc by definition.

A peer *blindly* submits requests to other peers which will in turn search for the requested resource. This can make the search time become unbounded. In addition, the lookup process can fail even if the resource exists, thus making the system be non-deterministic.

More recent systems like CAN, Chord ([Stoica 2001],[Castro 2002]), Oceanstore

[Kubiatowicz 2000] and PAST [Druschel 2001], impose a rigorous matching between the resource's identifier and the node providing that resource. Thus, a resource can always be retrieved as long as the hosting machines can be reached. Nodes, in those systems, become a network layer. Each node contains information about a small number of other nodes of the system. This limits the number of states to update if the list of resources changes through time and improves extensibility/scalability. The logical topology of this new network layer guarantees some properties about the cost of request processing. Oceanstore had been designed to extend to billions of users, millions of servers and more than 10^{14} files [Kubiatowicz 2000].

A.1.4 Anonymity

One of the usage made of the Peer-to-Peer technology is to allow individuals to use communication systems, without any sort of legal (or other type) constraints, on one hand, and to guarantee that no censorship of digital content is possible. The designers of Free Haven [Dingledine 2001] identified the following elements related to anonymity :

- Author : The author or the creator of a document can not be identified.
- Publisher : The individual that published the document on the system can not be identified.
- Reader : People that read or use content can not be identified.
- Server : Servers hosting a document can not be identified through this document.
- Document : Servers do not keep any knowledge about the content they host.
- Request : A server can not say which document is used to satisfy somebody's request.

In addition to these points , there exist three different types of anonymity between each communicating peer :

- Sender anonymity, masking sender's identity
- Receiver anonymity, masking receiver's identity
- Mutual anonymity, where both sender and receiver are hidden to each other as well as other peers[Pfitzmann 1987].

It is also crucial to determine the degree of anonymity that can be reached by a given method. Reiter and Rubin [1998] propose an overview of the different degrees of anonymity that ranges among "*absolute intimacy*", "*possible exposure*" and "*definitely exposed*" Absolute intimacy means that , even if an attacker can

determine that a message has been sent, the message's sender can not be determine with absolute probability.

There exist six methods among the most commonly used, adapted to different degrees of anonymity under various constraints.

A.1.4.1 Multicasting

. Multicasting (or broadcasting) can be used to ensure receiver's anonymity [Pfitzmann and Waidner 1987].

A multicast group is created for users willing to remain anonymous. A user wanting to get a document registers to the corresponding group. The user inside the group which holds the document, sends it to the whole group. The receiver's identity is then effectively hidden for the other peers and sender's anonymity is guaranteed. This technique takes advantage from the underlying network that has the multicast feature, like Ethernet or Token Ring.

A.1.4.2 Mask sender's identity

In non-connected protocols like UDP, sender's anonymity can be guaranteed by spoofing the sender's IP adress. However, this requires to change protocol. Besides, it can not always be achieved as most ISPs keep trace of network packets originating from invalid IP adresses.

A.1.4.3 Spoof identity.

Instead of changing the sender's identity, anonymity can be achieved by modifying the identity of an interlocutor. For instance, in Freenet [Clarke 2001], a peer sending data to another peer can claim itself as the data's owner. Thus, an attacker can not be sure that the sender of a data is the actual sender.

A.1.4.4 Stealth paths.

Instead of communicating directly, two entities can do so through intermediate nodes. Most of existing techniques only ensure sender's anonymity. A peer willing to hide its identity elaborates a stealth path towards its interlocutor. One can cite Mix , Onion [Syverson 1997], Anonymizing Proxy [Gabber 1999], Crowds [Reiter 1998] and Herdes [Shields 2000]. Stealth paths can use persistent connections or buffered connections. By varying the path's length and by changing the path with a variable frequency, multiple degrees of anonymity can be achieved.

A.1.4.5 Untraceable aliases.

LPWA [Gabber 1999] is a proxy server that generates untraceable aliases for client machines. The client can create an account and be identified through it, thus masking its true identity from the server. This kind of method ensure sender's anonymity and make use of *reliable* proxies. The degree of anonymity is almost absolute.

A.1.4.6 Unwanted hosting.

Another interesting approach is to provide anonymity by hosting non-voluntarily a document on a node, using for instance hash functions. As the hosting is non-determinist, host can not be held responsible for hosting a document.

Now, we sum up the different forms of anonymity used into the most popular P2P networks as well as the techniques employed.

Gnutella [2001] and Freenet [Clarke 2001] provide anonymity through the way peers exchange documents. In Gnutella, a request is broadcast again and again until it reaches a peer hosting the requested document. In Freenet, a request is sent to peers that have the highest probability of hosting the requested document. The reply is sent along the same path.

APFS [Scarlata 2001] deal with mutual anonymity by considering that reliable centralization is impossible. Peers must inform a coordinator that they can act as directories. Both the sender and the receiver of a message must build up stealth paths.

Free Haven [Dingledine 2001] and Publius [Waldman 2000] are designed to guarantee protection against censorship. Document's anonymity is reinforced by splitting it afterwards and saved on multiple servers. Thus, a given server never hosts all the necessary data for external attackers. Mutual anonymity between the publisher and the reader is ensured through stealth paths. Both platforms build these paths by sending many anonymous emails. Publius could be improved by integrating reader's anonymity. Anonymous emails could be used also for publishing.

PAST [Rowstron 2001b], CAN [Ratnasamy 2001] and Chord [Stoica 2001] constitute a class of P2P systems based on a reliable infrastructure. One common characteristic is that object hosting is non-deterministic and a node can not be held responsible for hosting that object. The included routing mechanisms can be easily used to build stealth paths and ensure mutual anonymity.

A.1.5 Auto-organisation

In cybernetics, self-organization is defined as "a process where a system's organization increases spontaneously, that is it increases without being controlled by its environment or any external system". [Heylighen 1997].

In P2P systems, self-organization is necessary for extensibility (scalability), fault tolerance, resources volatility and infrastructure cost. The scale of P2P systems can not be predicted according to the number of systems involved, the number of users or the workload. Predicting one of these parameters can be very challenging without centralizing partially the system. Scalability intrinsically induces a higher probability of failure, which requires self-maintenance abilities from the system.

A similar reasoning can be applied to resources volatility. It can be challenging for a system configuration to remain as it is for a long period of time. Adaptation is necessary to face the departure or the joining of peers into the Peer-to-Peer network. As it would be to maintain a dedicated equipment and/or hire staff to manage such a dynamic environment, the issue is dealt with directly by the peers.

Some systems and products deal with self-organization. In Oceanstore [Bindel 2002, Kubiatowicz 2000], self-organization is applied to data localization and routing infrastructure [Kubiatowicz 2000],[Rhea 2001], [Zhao 2001]. Because of the sporadic availability of peers as well as variations of network latency and bandwidth, the infrastructure continuously adapts its routing and localization mechanisms.

In Pastry, [Rowstron 2001a], self-organization is managed through protocols dedicated to a node's departure or joining and based on a Fault-Tolerant network layer. Clients' requests are routed within $\lceil \log_{16} N \rceil$ hops.

FastTrack assigns faster searches and transfers to distributed self-organized networks. In these networks, the most powerful machines automatically become Superpeers and act as content directories, under criteria related to processing capacity and network communication (for instance, latency time and bandwidth).

SearchLing uses self-organization to adapt the network according the nature of search requests, allowing to reduce network traffic and the number of unsatisfied requests.

A.1.6 Ad-hoc connectivity

The ad-hoc nature of connectivity has a major impact on all classes of P2P systems. In the field of distributed computing, parallel applications can not be run on every system indefinitely. This availability may vary from one system to another. P2P systems and distributed applications must take into account this ad-hoc nature and be able to manage the departure or joining of computational resources in the network. Although it can be considered as an exceptional event in distributed systems, it is seen as usual or frequent in P2P systems.

In P2P applications dedicated to content sharing, users expect to have access to content whenever they want, regarding the connectivity of content providers. In the most reliable systems, with guarantees on quality of service, the ad-hoc characteristic is diminished through redundant service providers, but some of them may remain unavailable.

In collaborative P2P systems and applications, the ad-hoc nature of connectivity becomes more obvious. Users cooperating with each other are more and more

interested in mobile devices, thus making them more connected to Internet and better suited for collaborative work. To face this situation, cooperative systems allow transparent communication delays from disconnected systems. this can be achieved through the use of proxies dedicated to message reception , or other types of relays which can temporarily suspend communications for a disconnected system.

Besides, not all the systems will be connected to Internet. Even if it were the case, ad-hoc groups of users must be able to create ad-hoc networks to cooperate. Existing infrastructures like 802.11b/g, Bluetooth and Infrared. Thus, P2P systems and applications must be designed to support frequent exits or entries of peers.

A.1.7 Performances and security.

When designing a P2P system, those two points have a great importance regarding extensibility/scalability of an application.

A.1.7.1 Intelligent routing and network topology.

To take advantage of the great potential of P2P networks, it is crucial to understand and explore the possible interactions between peers. One of the most advanced works about social interactions is the "small-world phenomenon" on the Milgram experiment [1967]. The goal of that experiment was to find a series of acquaintances (that is, two people who know each other) that could link any couple of people in the US who do not know each other. By using postcards, Milgram discovered that, in the 1960's, Americans were linked through a "path" of six people in average. Adamic *et al.* explored power-law distributions for P2P networks, and introduced local search techniques using nodes having a high degree and a scalability cost sub-linear with the network size[Adamic 1999].

Ramanathan *et al.* [Ramanathan 2002] determine "good" peers, based on interests and manipulate dynamically network connections between peers to guarantee that high degree nodes with similar interests are strongly linked to one another. Establishing a good set of peers reduces the number of broadcast messages on the network and the number of peers processing a request before the result is found. Some academic systems like Oceanstore and Pastry, improve performances by moving data inside the network in a proactive way. The main advantage of these approaches is that peers can choose who they connect to and when to open or close a connection , based only on local information.

A.1.7.2 Firewalls.

P2P applications naturally require direct connections between peers. However, in some administrative domains, internal networks are isolated from the external world, granting only restricted access to applications. For instance, most firewalls block incoming TCP connections. That is, a machine located behind a firewall shall not

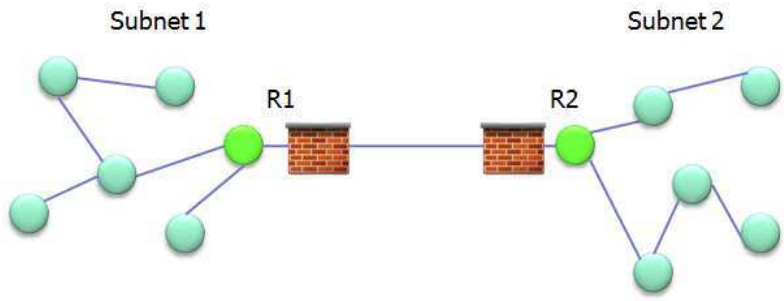

Figure A.2: Using relays to cross firewalls

be accessible from outside the network. Worse, individuals often use, at home, a private IP address behind a router or a NAT (Network Address Translation) service to share an Internet connection between multiple machines. This leads to the issue of non-accessibility. However, as outgoing connections through port 80 (HTTP) are often allowed by firewalls, some mechanisms have been designed to allow communications between hidden machines (behind a firewall or a NAT system, unreachable from Internet). This procedure is somewhat limited as it requires the connection to be initiated by the "hidden" machine. When other communicating peers are located between different firewalls, the issue becomes even more complex. A "reflecting" server on Internet will be necessary to allow connections between hidden peers.(Figure A.2).

A.1.8 Fault Tolerance

One of the main characteristics of a P2P network is to avoid having any central weak points. Although most of (pure) P2P networks already do it, they still face some failures which are commonly linked to systems involving multiple hosts or networks : disconnections, unreachable hosts, network partitioning and nodes crashes. These failures can be more significant in some networks (for instance, wireless networks) than others (corporate cabled networks). It would be more interesting to pursue a collaborative work between peers still connected to the network when facing such failures. One relevant example could be an application, like Genome@Home [2001] running a distributed algorithm on multiple peers. Is it still possible to continue calculations if one of the peers disappears because of a network connection failure ? Similar questions have to be answered to by P2P systems willing to provide more than a "best-effort" Internet service.

In the past, client-server disconnections were studied for distributed filesystems including mobile devices (for instance, Coda [Satyanarayanan 1990]) , and one solution would be to have solvers proper to each application in order to deal with inconsistences after reconnections. Some recent P2P networks (Groove [Groove 2001],[Microsoft 2008]) define special nodes, called "relays", that temporarily memorize any update or communication until the receiver (here, another

peer) reappears in the network. Others, like Magi [Bolcer 2000], delay messages at the source, until the receiving peer is detected.

A disconnection can result from a resources unavailability. This can occur when a machine becomes unreachable due to a network failure, or when the peer holding the resource crashes. Whereas the first case can be solved by routing information through an alternate path (a feature already provided by Internet), the second one requires greater attention. Crucial resources duplication can simplify the problem. Networks like Napster and Gnutella are systems allowing passive and uncontrolled duplication of data, based only on date's popularity. Depending on the application being run on these networks, it can be necessary to implement data persistency, by proposing a reliable data duplication policy.

Anonymous broadcasting services like Freenet [Clarke 2001] and Publius [Waldman 2000] ensure data availability through controlled replication. Oceanstore [Kubiatowicz 2000] introduces a system with two replication layers, and, through monitoring administrative domains, avoids to send replicas to sites with a high probability of failure. However, as a resource into a P2P network can be more than just a file (like proxies on Internet, online storage space, or shared computational power), the concepts of distributed data replication must be extended to other types of resources. Some solutions for distributed computing on grids (for instance, Legion [Grimshaw 1997]) propose fault tolerance for nodes by restarting calculations on other nodes.

One of the main challenges for a P2P system is that it manages its own maintenance, a task distributed over all the peers, to ensure resources availability. This is different from client-server systems, where resources availability is ensured by the server.

A.2 Existing Implementations

In this section, we propose an overview of the main existing P2P networks : Avaki, SETI@Home, Groove, Magi, FreeNet, Gnutella, JXTA, .NET, NaradaBrokering, Pastry.

Avaki [Avaki 2002] is designed to allow to see a network of heretogeneous resources as a single virtual machine. This is a classical example of "*meta-computing*" which is applied to networks ranging from corporate systems and data grids to global computational grids over Internet. This is an object-oriented system. Each entity is considered as an addressable object with a set of methods and interfaces. It is mostly inspired from Mentat [Grimshaw 1993].

SETI (Search for ExtraTerrestrial Intelligence)) is a collection of projects aiming to discover extraterrestrial civilizations. One of these projects, SETI@Home,

analyzes radio signals received from space and collected by the giant telescope in Arecibo, by using the computing power of millions of unused machines. [Anderson 2002].

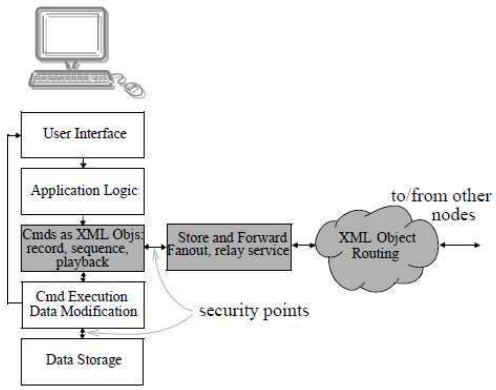

Figure A.3: Groove Architecture

Groove [Microsoft 2008] is a collaborative P2P system, which can be also considered as a platform. Groove mainly addresses to Internet users as well as intranet ones. It can also be used on mobile devices like PDAs, mobile phones or touchscreen tablets (Figure A.3). It enables communications, content sharing and proposes tools for joint activities. For instance, all Groove applications inherit from the integrated security mechanism without the need to redefine algorithms linked to security. This allows the deployment of safe applications.

Magi [Bolcer 2000] is a P2P infrastructure for designing secure, portable and collaborative applications. It uses standardized protocols like HTTP, WebDAV and XML to allow communications between applications inside corporate networks or Internet (Figure A.4). Magi Enterprise, the final product, builds an infrastructure which links computers of project teams to enable file sharing, instant messaging ....

FreeNet [Freenet 2001] is a P2P file sharing systems based on a model proposed by Ian Clarke [Clarke 2001]. The prime objective of FreeNet is to guarantee user's anonymity. That is, when logging into the system, a user must be able to submit requests without anyone discovering sender's identity. a FreeNet user has no knowledge about the content stored on its hard disk.
Gnutella [Osokine 2002] is a file sharing protocol. Applications implementing Gnutella allow users to search and download content from other users connected to Internet.

The objective of the **JXTA** project [Daniel Brookshier 2002] is to propose an open and innovative collaboration platform which supports a wide range of distributed computing applications and allows to harness computing power from any device connected to the network [Gong 2001]. JXTA provides features on multiple layers, including basic mechanisms and concepts, high-level services that

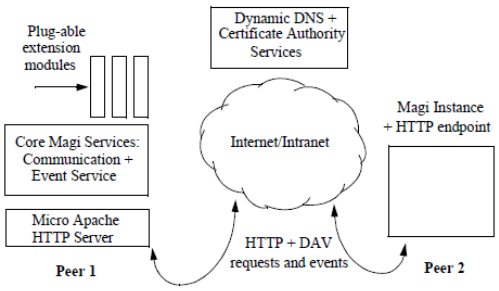

Figure A.4: Magi Architecture

extend these mechanisms to a wide range of applications, which demonstrates the platform's potential.

NaradaBrokering [Fox 2005],[Pallickara 2003] is a technology that takes advantage from two distributed application systems : P2P systems and grid-based systems. Both systems have complementary advantages. This technology can be interfaced with a JXTA P2P network. The main idea behind NaradaBrokering is to provide users with a high quality network. To do so, NaradaBrokering uses techniques from distributed objects, web services and message-oriented middlewares. The architecture used is made of local networks of peers that are linked at large scale through a "core" interconnection network. To access the services, "brokers" are introduced. Messages-oriented middlewares are mainly used on this part of the network. Brokers are considered as the smallest unit of this network. They are in charge of processing and routing messages intelligently. The NaradaBrokering network is considered as a network made of brokers which cooperate with one another, whether they are located on client machines or not. To avoid having a loosely connected network, NaradaBrokering integrates a inter-broker communication protocol to manage links between brokers and manage the addition (or substraction) of brokers into/from the network. The network's organization is hierarchical, each broker belonging to a sub-group, forming greater groups with other sub-groups and so on. These first sub-groups have strongly interconnected brokers to guarantee at least one link even in case of failure. Thus, we have many small networks connected to one another. Network traffic increases by a logarithmic trend and not exponential like disorganized networks. NaradaBrokering also introduces Brokers Network Maps (BNM) that allow the intelligent routing feature described here above.

Pastry [Gendron 1994],[Castro 2002], [Rowstron 2001a] is a P2P system whose main characteristic is to propose a topology and a routing algorithm that allows to have a network traffic increasing logarithmically with the number of peers and avoids the definition of another type of peers as in NaradaBrokering.

Tapestry [Zhao 2001] is very similar to Pastry, the only difference being a variant in the routing algorithm. In our works, we inspired ourselves from Pastry, which we detail more thoroughly.

Publications

Book Chapters

- **M. Djamaï**, B. Derbel and N. Melab, "***Large Scale P2P-Inspired Problem Solving: A Formal and Experimental Study***", Large Scale Network-Centric Computing Systems, Wiley, 2012, *To be published.*

International Conferences

- **M. Djamaï**, B. Derbel and N. Melab, "***Impact of overlay properties upon a P2P approach for parallel B&B***", The 1st International Conference on Systems and Computer Science, August 29^{th}-31^{st} 2012, Villeneuve d'Ascq, France.

- **M. Djamaï**, B. Derbel and N. Melab, "***A Large-Scale Pure P2P approach for the B&B algorithm***", The Fourth IEEE International Scalable Computing Challenge (SCALE 2011) held in conjunction with The 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'2011), Newport Beach, USA, May 23^{rd}-26^{th}, 2011.

International Workshops

- **M. Djamaï**, B. Derbel and N. Melab, "***Distributed B&B : A Pure Peer-to-Peer Approach***", In Proceedings of the 25^{th} IEEE International Parallel and Distributed Processing Symposium (IPDPS) – Workshop on Large-Scale Parallel Processing (LSPP'11), Anchorage, Alaska, USA May 16^{th}-20^{th}, 2011.

- **M. Djamaï**, B. Derbel and N. Melab, "***Experimental Study of a P2P B&B approach on top of Grid'5000***",In Grid'5000 Workshop and Spring School, Reims, France, April 18^{th}-21^{st}, 2011.

- **M. Djamaï**, B. Derbel and N. Melab, "***Distributed Branch-and-Bound Algorithm : A Pure Peer-to-Peer Approach***" , In Grid'5000 Workshop and Spring School, Lille, France, April 6^{th}-9^{th}, 2010.

Bibliography

[Abadi 2000] I.N. Kamal Abadi, Nicholas G. Hall and Chelliah Sriskandarajah. *Minimizing Cycle Time in a Blocking Flowshop.* Operations Research, vol. 48, no. 1, pages 177–180, January/February 2000. (Not cited.)

[Adamic 1999] Lada A. Adamic. *The Small World Web.* In 3^{th} European Conference on Research and Advanced Technology for Digital Libraries (ECDL'99), pages 443–452, London, UK, 1999. Springer-Verlag. (Not cited.)

[Aida 2002] K. Aida and Y. Futakata. *High-performance parallel and distributed computing for the BMI eigenvalue problem.* In 16^{th} International Parallel and Distributed Processing Symposium, (IPDPS'02), pages 71 –78, 2002. (Not cited.)

[Aida 2005] K. Aida and T. Osumi. *A Case Study in Running a Parallel Branch and Bound Application on the Grid.* In SAINT '05: The 5^{th} IEEE Symposium on Applications and the Internet, pages 164–173, Washington, DC, USA, jan. 2005. (Not cited.)

[Allahverdi 1999] Ali Allahverdi, Jatinder N.D Gupta and Tariq Aldowaisan. *A review of scheduling research involving setup considerations.* Omega, vol. 27, no. 2, pages 219 – 239, 1999. (Not cited.)

[Allahverdi 2004] Ali Allahverdi and Tariq A. Aldowaisan. *No-wait flowshops with bicriteria of makespan and maximum lateness.* European Journal of Operational Research, pages 132–147, 2004. (Not cited.)

[Anderson 2002] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky and Dan Werthimer. *SETI@home: an experiment in public-resource computing.* Communications of the ACM, vol. 45, no. 11, pages 56–61, November 2002. (Not cited.)

[Andrés 2005] Carlos Andrés, José Miguel Albarracin, Guillermina Tormo, Eduardo Vicens and José Pedro Garcia-Sabater. *Group technology in a hybrid flowshop environment: A case study.* European Journal of Operational Research, vol. 167, no. 1, pages 272 – 281, 2005. (Not cited.)

[Avaki 2002] Avaki. Avaki grid software: Concepts and architecture. Using Comprehensive Grid Software from AVAKI to Provide Wide-Area Access to Processing Power, Applications, and Data, Mars 2002. (Not cited.)

[Awerbuch 1985] B. Awerbuch. *Complexity of network synchronization.* Journal of the ACM, vol. 32, no. 4, pages 804–823, 1985. (Not cited.)

[Bakken 2002] Dave Bakken. Paradigms for distributed fault tolerance, Février 2002. (Not cited.)

[Barabasi 1999] Albert-Laszlo Barabasi and Reka Albert. *Emergence of Scaling in Random Networks*. Science, vol. 286, no. 5439, pages 509–512, 1999. (Not cited.)

[Barrat 2000] A. Barrat and M. Weigt. *On the properties of small-world network models*. The European Physical Journal B, Volume 13, Issue 3, pp. 547-560 (2000)., vol. 13, pages 547–560, January 2000. (Not cited.)

[Bendjoudi 2009] A. Bendjoudi, N. Melab and E.-G. Talbi. *P2P design and implementation of a parallel branch and bound algorithm for grids*. International Journal of Grid and Utility Computing, vol. 1, no. 2, pages 159–168, 2009. (Not cited.)

[Bendjoudi 2011] Ahcene Bendjoudi, Nouredine Melab and El-Ghazali Talbi. *Fault-Tolerant Mechanism for Hierarchical Branch and Bound Algorithm*. 2011. (Not cited.)

[Bendjoudi 2012] Ahcène Bendjoudi. *Scalable and Fault-Tolerant Hierarchical B&B Algorithms For Computational Grids*. PhD thesis, Université A.MIRA-BEJAIA Faculté des Sciences Exactes Département Informatique, 2012. (Not cited.)

[Bendjoudi 2013] A. Bendjoudi, N. Melab and E-G. Talbi. *FTH-B&B: a Fault-Tolerant Hierarchical Branch and Bound for Large Scale Unreliable Environments*. IEEE Transactions on Computers, 2013. (Not cited.)

[Bindel 2002] David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, Ben Zhao and John Kubiatowicz. *OceanStore: An Extremely Wide-Area Storage System*. Rapport technique, Berkeley, CA, USA, 2002. (Not cited.)

[Bittorrent 2005] Bittorrent. *Bittorrent Protocol Specification*, 2005. (Not cited.)

[Blumofe 1996] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall and Yuli Zhou. *Cilk: An Efficient Multithreaded Runtime System*. Journal of Parallel and Distributed Computing, vol. 37, no. 1, pages 55–69, August 25 1996. (Not cited.)

[Bolcer 2000] G. A. Bolcer. *Magi: an architecture for mobile and disconnected workflow*. vol. 4, no. 3, pages 46–54, May–June 2000. (Not cited.)

[Bonney 1976] M C Bonney and S W Gundry. *Solutions to the constrained flowshop sequencing problem*. Operational Research Quarterly, no. 27, page 869, 1976. (Not cited.)

[Cabani 2007] A. Cabani, S. Ramaswamy, M. Itmi and J.-P. Pécuchet. *PHAC: An Environment for Distributed Collaborative Applications on P2P Networks*. In

6th International Conference on Distributed Computing and Internet Technology (ICDCIT), pages 240–247, 2007. (Not cited.)

[Cappello 1997] Peter Cappello, Bernd Christiansen, Mihai F. Ionescu, Michael O. Neary, Klaus E. Schauser and Daniel Wu. *Javelin: Internet-Based Parallel Computing Using Java*, 1997. (Not cited.)

[Caromel 2007] Denis Caromel, Alexandre di Costanzo, Laurent Baduel and Satoshi Matsuoka. *Grid'BnB: a parallel branch and bound framework for grids.* In Proceedings of the 14th international conference on High performance computing, HiPC'07, pages 566–579, Berlin, Heidelberg, 2007. Springer-Verlag. (Not cited.)

[Castro 2002] Miguel Castro, Peter Druschel, Y. Charlie Hu and Antony Rowstron. *Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks*, 2002. (Not cited.)

[Chandra 2000] R Chandra, R.M. Lefever, M Cukier and W.H. Sanders. *Loki: a state-driven fault injector for distributed systems.* In Proceeding International Conference on Dependable Systems and Networks. DSN 2000, pages 237–242. IEEE Comput. Soc, 2000. (Not cited.)

[Clarke 2001] Ian Clarke, Oskar Sandberg, Brandon Wiley and Theodore W. Hong. *Freenet: A Distributed Anonymous Information Storage and Retrieval System.* Lecture Notes in Computer Science, vol. 2009, pages 46–50, 2001. (Not cited.)

[Claudel 2009] Benoit Claudel, Guillaume Huard and Olivier Richard. *TakTuk, adaptive deployment of remote executions.* In Proceedings of the 18th ACM international symposium on High performance distributed computing, HPDC '09, pages 91–100, New York, NY, USA, 2009. ACM. (Not cited.)

[Cung 1994] Dowaji Cung, Mautor Le Cun and Chris Roucairol. *Parallel and distributed branch-and-bound/A* algorithms.* Rapport technique, PRiSM Laboratory, Technical Report n94/31, Octobre 1994. (Not cited.)

[Dakin 1965] R. J. Dakin. *A tree-search algorithm for mixed integer programming problems.* The Computer Journal, vol. 8, no. 3, pages 250–255, March 1965. (Not cited.)

[Daniel Brookshier 2002] Brendon Wilson Daniel Brookshier Sing Li. *JXTA : P2P Grows Up.* Java Sun Technical Articles, vol. 1, pages 1–6, December 2002. (Not cited.)

[Dawson 1996] Scott Dawson, Farnam Jahanian and Todd Mitton. *ORCHESTRA: A fault injection environment for distributed systems.* Ann Arbor, pages 1–30, 1996. (Not cited.)

[Di Constanzo 2007] A. Di Constanzo. *Branch-and-bound with peer-to-peer for large-scale grids*. PhD thesis, Ecole doctorale STIC, Sophia Antipolis, France, Octobre 2007. (Not cited.)

[Dijkstra 1980] Edsger W. Dijkstra and C. S. Scholten. *Termination detection for diffusing computations*. Information Processing Letters, vol. 11, no. 1, pages 1–4, August 1980. (Not cited.)

[Dingledine 2001] Freedman M. Rubin A. Dingledine R. Peer-to-peer. harnessing the power of disruptive technologies, chapitre Free Haven, pages 159–187. Oram A., 2001. (Not cited.)

[Drummond 2006] Lúcia M.A. Drummond, Eduardo Uchoa, Alexandre D. Gonçalves, Juliana M.N. Silva, Marcelo C.P. Santos and Maria Clícia S. de Castro. *A grid-enabled distributed branch-and-bound algorithm with application on the Steiner Problem in graphs*. Parallel Computing, vol. 32, no. 9, pages 629–642, October 2006. (Not cited.)

[Druschel 2001] Peter Druschel and Antony I. T. Rowstron. *PAST: A large-scale, persistent peer-to-peer storage utility*. pages 75–80, 2001. (Not cited.)

[Eckstein 2000] Jonathan Eckstein, Jonathan Eckstein, Cynthia A. Phillips, Cynthia A. Phillips, William E. Hart and William E. Hart. *PICO: An Object-Oriented Framework for Parallel Branch and Bound*. Rapport technique, Rutgers University, Piscataway, NJ, 2000. (Not cited.)

[Erdös 1959] P. Erdös and A. Rényi. *On random graphs, I.* Publicationes Mathematicae (Debrecen), vol. 6, pages 290–297, 1959. (Not cited.)

[Erdos 1960] P. Erdos and A. Renyi. *On the evolution of random graphs*. Publ. Math. Inst. Hung. Acad. Sci, vol. 5, pages 17–61, 1960. (Not cited.)

[Fedak 2003] Gilles Fedak. *XtremWeb : une plate-forme pour l'étude expérimentale du calcul global pair-à-pair*. PhD thesis, Université Paris XI, 2003. (Not cited.)

[Finkel 1987] Raphael Finkel and Udi Manber. *DIB—a distributed implementation of backtracking*. ACM Trans. Program. Lang. Syst., vol. 9, no. 2, pages 235–256, 1987. (Not cited.)

[Foster 1997] Ian Foster and Carl Kesselman. *Globus: A Metacomputing Infrastructure Toolkit*. International Journal of Supercomputer Applications, vol. 11, pages 115–128, 1997. (Not cited.)

[Fox 2005] G. Fox and S. Pallickara. *Deploying the NaradaBrokering Substrate in Aiding Efficient Web and Grid Service Interactions*. vol. 93, no. 3, pages 564–577, March 2005. (Not cited.)

[Freenet 2001] Project Freenet. *Understand Freenet*, 2001. (Not cited.)

[Frey 2002] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster and Steven Tuecke. *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* Cluster Computing, vol. 5, no. 3, pages 237–246, 2002. (Not cited.)

[Gabber 1999] Eran Gabber, Phillip B. Gibbons, David M. Kristol, Yossi Matias and Alain Mayer. *Consistent, yet anonymous, Web access with LPWA.* Commun. ACM, vol. 42, no. 2, pages 42–47, 1999. (Not cited.)

[Gangadharan 1993] Rajesh Gangadharan and Chandrasekharan Rajendran. *Heuristic algorithms for scheduling in the no-wait flowshop.* International Journal of Production Economics, vol. 32, no. 3, pages 285–290, 1993. (Not cited.)

[Gendron 1994] B. Gendron and T. G. Crainic. *Parallel Branch-And-Bound Algorithms : Survey and Synthesis.* Operations Research, vol. 42, no. 6, pages 1042–1066, Nov-Dec 1994. (Not cited.)

[Grid'5000] Grid'5000. *Grid'5000 Website.* (Not cited.)

[Grimshaw 1997] Andrew S. Grimshaw, Wm. A. Wulf and CORPORATE The Legion Team. *The Legion vision of a worldwide virtual computer.* Commun. ACM, vol. 40, no. 1, pages 39–45, 1997. (Not cited.)

[Groove 2001] Networks Groove. *Groove Networks Prodict Backgrounder.* 2001. (Not cited.)

[Hall 1996] Nicholas G. Hall and Chelliah Sriskandarajah. *A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process.* Operations Research, vol. 44, no. 3, pages 510–525, May/June 1996. (Not cited.)

[Huffaker 2002] Bradley Huffaker, Marina Fomenkov, Daniel J. Plummer, David Moore, K. claffy, Bradley Huffaker Marina Fomenkov and A. Background. *Distance Metrics in the Internet.* In in IEEE International Telecommunications Symposium, pages 200–2, 2002. (Not cited.)

[Iamnitchi 2000] A. Iamnitchi and I. Foster. *A problem-specific fault-tolerance mechanism for asynchronous, distributed systems.* In Proceedings of the 29^{th} International Conference on Parallel Processing, pages 4–13, 21–24 Aug. 2000. (Not cited.)

[Johnson 1954] S. M. Johnson. *Optimal two- and three-stage production schedules with setup times included.* Naval Research Logistics Quarterly, vol. 1, no. 1, pages 61–68, 1954. (Not cited.)

[KING 1980] J. R. KING and A. S. SPACHIS. *Heuristics for flow-shop scheduling.* International Journal of Production Research, vol. 18, no. 3, pages 345–357, 1980. (Not cited.)

[Kleinberg 2000a] Jon Kleinberg. *The Small-World Phenomenon: An Algorithmic Perspective*. pages 163–170, 2000. (Not cited.)

[Kleinberg 2000b] Jon M Kleinberg. *Navigation in a small world*. Nature, vol. 406, no. 6798, page 845, 2000. (Not cited.)

[Kubiatowicz 2000] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells and Ben Zhao. *OceanStore: an architecture for global-scale persistent storage*. SIGPLAN Not., vol. 35, no. 11, pages 190–201, 2000. (Not cited.)

[Kumar 1984] V. Kumar and L. N. Kanal. *Parallel Branch-and-Bound Formulations for And/or Tree Search*. IEEE Transactions on Pattern Analysis and Machine Intelligence,, vol. PAMI-6, no. 6, pages 768 –778, nov. 1984. (Not cited.)

[Lageweg 1978] B. J. Lageweg, J. K. Lenstra and A. H. G. Rinnooy Kan. *A General Bounding Scheme for the Permutation Flow-Shop Problem*. Operations Research, vol. 26, no. 1, pages 53–67, 1978. (Not cited.)

[Lalami 2012] Mohamed Esseghir Lalami and Didier El-Baz. *GPU Implementation of the Branch and Bound Method for Knapsack Problems*. 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, vol. 0, pages 1769–1777, 2012. (Not cited.)

[Land 1960] AH Land and AG Doig. *An automatic method of solving discrete programming problems*. Econometrica: Journal of the Econometric Society, vol. 28, no. 3, pages 497–520, 1960. (Not cited.)

[Little 1963] G. Little, K. Murty, D. Sweyney and C. Karel. *An algorithm for the travelling salesman problem*. In Operations Research, 1963. (Not cited.)

[Litzkow 1988] M. J. Litzkow, M. Livny and M. W. Mutka. *Condor-a hunter of idle workstations*. In Proc. th International Conference on Distributed Computing Systems, pages 104–111, 13–17 June 1988. (Not cited.)

[Mans 1995] Bernard Mans, Thierry Mautor and Catherine Roucairol. *A parallel depth first search branch and bound algorithm for the quadratic assignment problem*. European Journal of Operational Research, vol. 81, no. 3, pages 617 – 628, 1995. (Not cited.)

[Mattern 1987] Friedemann Mattern. *Algorithms for Distributed Termination Detection*. Distributed Computing, vol. 2, no. 3, pages 161–175, 1987. (Not cited.)

[Maymounkov 2002] P. Maymounkov and D. Mazieres. *Kademlia: A peer-to-peer information system based on the XOR metric*. 2002. (Not cited.)

[Mehdi 2011] Malika Mehdi. *Parallel hybrid optimization methods for permutation based problems*. PhD thesis, Université Lille1 - Sciences et Technologies, Université du Luxembourg, 2011. (Not cited.)

[Melab 1996] Nordine Melab, Nathalie Devesa, Marie-Paule Lecouffe and Bernard Toursel. *Adaptive Load Balancing of Irregular Applications - A Case Study: IDA* Applied to the 15-Puzzle Problem*. In IRREGULAR '96: Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems, pages 327–338, London, UK, 1996. Springer-Verlag. (Not cited.)

[Melab 1997] Nouredine Melab. *Gestion de la granularité et régulation de charge dans le modèle P3 d'évaluation parallèle des langages fonctionnels*. PhD thesis, Lille 1, Grenoble, 1997. Th. : informatique. (Not cited.)

[Melab 2005] Nouredine Melab. *Contributions à la résolution de problèmes d'optimisation combinatoire sur grilles de calcul*, Novembre 2005. HDR. (Not cited.)

[Melab 2012] Nouredine Melab, Imen Chakroun, Mohand-Said Mezmaz and Daniel Tuyttens. *A GPU-accelerated Branch-and-Bound Algorithm for the Flow-Shop Scheduling Problem*. CoRR, vol. abs/1208.3933, 2012. (Not cited.)

[Mezmaz 2005] M. Mezmaz, Melab N. and E.-G Talbi. *Towards a Coordination Model for Parallel Cooperative P2P Multi-objective Optimization*. In European Grid Conference (EGC'2005), pages 305–314, 2005. (Not cited.)

[Mezmaz 2007a] M. Mezmaz. *Une approche efficace pour le passage sur grilles de calcul de méthodes d'optimisation combinatoire*. PhD thesis, Université des Sciences et Technologies de Lille 1, Novembre 2007. (Not cited.)

[Mezmaz 2007b] M. Mezmaz, N. Melab and E.-G. Talbi. *A Grid-enabled Branch and Bound Algorithm for Solving Challenging Combinatorial Optimization Problems*. In 21st International Parallel and Distributed Processing Symposium, 2007. (IPDPS'07)., pages 1–9, March 2007. (Not cited.)

[Microsoft 2008] Microsoft. *Groove Protocols Review*, Décembre 2008. (Not cited.)

[Miller 1993] D.L. Miller and J.F. Pekny. *The Role of Performance Metrics for Parallel Mathematical Programming Algorithms*. In ORSA J. Computing, volume 5, pages 26–28, 1993. (Not cited.)

[Milojicic 2008] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu. *Peer-to-Peer Computing*. Rapport technique, 2008. (Not cited.)

[Mittal 2004] Neeraj Mittal, S. Venkatesan and Sathya Peri. *Message-Optimal and Latency-Optimal Termination Detection Algorithms for Arbitrary Topologies*.

In Proceedings of the 18th Symposium on Distributed Computing (DISC, pages 290–304, 2004. (Not cited.)

[Moran 2000] S. Moran and S. Snir. *Simple and efficient network decomposition and synchronization.* Theoretical Computer Science, vol. 243, no. 1-2, pages 217–241, 2000. (Not cited.)

[Nguyen 2012] The Tung Nguyen and Didier El Baz. *Fault Tolerant Implementation of Peer-to-peer Distributed Iterative Algorithms.* In Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on, pages 137 –145, dec. 2012. (Not cited.)

[Osokine 2002] S. Osokine. *Search Optimization in the Distributed Networks.* Internet, Octobre 2002. (Not cited.)

[Pallickara 2003] Shrideep Pallickara and Geoffrey Fox. *NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids.* In Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, pages 41–61, New York, NY, USA, 2003. Springer-Verlag New York, Inc. (Not cited.)

[Papadimitriou 1998] CH Papadimitriou and K Steiglitz. Combinatorial optimization: algorithms and complexity. 1998. (Not cited.)

[Pfitzmann 1987] A Pfitzmann and M Waidner. *Networks without user observability.* Comput. Secur., vol. 6, no. 2, pages 158–166, 1987. (Not cited.)

[Prieditis 1998] Armand Prieditis. *Depth-First Branch-and-Bound vs. Depth-Bounded IDA*.* Computational Intelligence, vol. 14, no. 2, pages 188–206, 1998. (Not cited.)

[Ralphs 2003] T.K. Ralphs, L. Ladanyi and M.J. Saltzman. *Parallel branch, cut, and price for large-scale discrete optimization.* Mathematical Programming, vol. 98, no. 1-3, pages 253–280, September 2003. (Not cited.)

[Ramanathan 2002] Murali Krishna Ramanathan, Vana Kalogeraki and Jim Pruyne. *Finding Good Peers in Peer-to-Peer Networks.* In IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium, page 158, Washington, DC, USA, 2002. IEEE Computer Society. (Not cited.)

[Ratnasamy 2001] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Schenker. *A scalable content-addressable network.* In SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, volume 31, pages 161–172. ACM Press, October 2001. (Not cited.)

[Reddi 1972] S. S. Reddi and C. V. Ramamoorthy. *On the Flow-Shop Sequencing Problem with No Wait in Process[dagger].* J Oper Res Soc, vol. 23, no. 3, pages 323–331, September 1972. (Not cited.)

[Reiter 1998] Michael K. Reiter and Aviel D. Rubin. *Crowds: anonymity for Web transactions.* ACM Trans. Inf. Syst. Secur., vol. 1, no. 1, pages 66–92, 1998. (Not cited.)

[Reza Hejazi 2005] S. Reza Hejazi and S. Saghafian. *Flowshop-scheduling problems with makespan criterion: a review.* International Journal of Production Research, vol. 43, no. 14, pages 2895–2929, 2005. (Not cited.)

[Rhea 2001] Sean Rhea, Chris Wells, Patrick Eaton, Dennis Geels, Ben Zhao, Hakim Weatherspoon and John Kubiatowicz. *Maintenance-Free Global Data Storage.* IEEE Internet Computing, vol. 5, no. 5, pages 40–49, 2001. (Not cited.)

[Röck 1984] Hans Röck. *The Three-Machine No-Wait Flow Shop is NP-Complete.* J. ACM, vol. 31, no. 2, pages 336–345, March 1984. (Not cited.)

[Rowstron 2001a] A. Rowstron and P. Druschel. *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.* In Lecture Notes in Computer Science, pages 329–350, 2001. (Not cited.)

[Rowstron 2001b] A. Rowstron and P. Druschel. *Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility*, 2001. (Not cited.)

[Saffre 2003] Fabrice Saffre and Robert Ghanea-Hercock. *Beyond anarchy: self-organized topology for peer to peer networks.* Complexity, vol. 9, no. 2, pages 49–53, 2003. (Not cited.)

[Sato 1997] Mitsuhisa Sato, Hidemoto Nakada, Satoshi Sekiguchi and Satoshi Matsuoka. *Ninf: A network based information library for global world-wide computing infrastructure.* pages 491–502, 1997. (Not cited.)

[Scarlata 2001] V. Scarlata, B.N. Levine and C. Shields. *Responder anonymity and anonymous peer-to-peer file sharing.* pages 272–280, Nov. 2001. (Not cited.)

[Seymour 2002] Keith Seymour, Hidemoto Nakada, Satoshi Matsuoka, Jack Dongarra, Craig Lee and Henri Casanova. *Overview of GridRPC: A Remote Procedure Call API for Grid Computing.* pages 274–278, 2002. (Not cited.)

[Shabtay 1994] Lior Shabtay and Adrian Segall. *Low Complexity Network Synchronization.* Proceedings of the 8th International Workshop on Distributed Algorithms.(WDAG '94), pages 223–237, 1994. (Not cited.)

[Shields 2000] Clay Shields and Brian Neil Levine. *A protocol for anonymous communication over the Internet.* In CCS '00: Proceedings of the 7th ACM

conference on Computer and communications security, pages 33–42, New York, NY, USA, 2000. ACM. (Not cited.)

[Stoica 2001] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan. *Chord: A scalable peer-to-peer lookup service for internet applications*. In SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pages 149–160, New York, NY, USA, 2001. ACM. (Not cited.)

[Stott 2000] D.T. Stott, B Floering, D Burke, Z Kalbarczpk and R.K. Iyer. *NFTAPE: a framework for assessing dependability in distributed systems with lightweight fault injectors*. In Proceedings IEEE International Computer Performance and Dependability Symposium. IPDS 2000, pages 91–100. IEEE Comput. Soc, 2000. (Not cited.)

[Stutzbach 2006] Daniel Stutzbach and Reza Rejaie. *Understanding churn in peer-to-peer networks*. Proceedings of the 6th ACM SIGCOMM on Internet measurement - IMC '06, page 189, 2006. (Not cited.)

[Syverson 1997] Paul F. Syverson, David M. Goldschlag and Michael G. Reed. *Anonymous Connections and Onion Routing*, 1997. (Not cited.)

[Taillard 1993] E. Taillard. *Benchmarks for basic scheduling problems*. European Journal of Operational Research, vol. 64, no. 2, pages 278 – 285, 1993. Project Management and Scheduling. (Not cited.)

[Tanaka 2003] Y Tanaka, H Nakada, S Sekiguchi, T Suzumura and S Matsuoka. *Ninf-G : A Reference Implementation of RPC-based Programming Middleware for Grid Computing*. Journal of Grid Computing, vol. 1, no. 1, pages 41–51, 2003. (Not cited.)

[Tanenbaum 2002] AS Tanenbaum and M Van Steen. Distributed systems: principles and paradigms. Prentice Hall PTR, 2002. (Not cited.)

[V.K. Janakiram 1988] D.P. Agrawal V.K. Janakiram and R. Mehrotra. *A Randomized Parallel Branch-and-Bound Algorithm*. In in Proc. of Int. Cont. on Parallel Processing, pages 69–75, Août 1988. (Not cited.)

[Waldman 2000] Marc Waldman, Aviel D. Rubin and Lorrie Faith Cranor. *Publius: a robust, tamper-evident, censorship-resistant web publishing system*. In SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium, pages 5–5, Berkeley, CA, USA, 2000. USENIX Association. (Not cited.)

[Watts 1998] Duncan J. Watts and Steven H. Strogatz. *Collective dynamics of "small-world" networks*. 1998. (Not cited.)

[Y. Xu 2005] L. Ladanyi M.-J. Saltzman Y. Xu T. K. Ralphs. *ALPS : A Framework for Implementing Parallel Search Algorithms.* pages 319–334, 2005. (Not cited.)

[Zhang 2000] Weixiong Zhang. *Depth-first branch-and-bound versus local search: A case study.* In In Proc. 17th National Conf. on Artificial Intelligence (AAAI-2000, pages 930–935, 2000. (Not cited.)

[Zhao 2001] Ben Y. Zhao, John D. Kubiatowicz and Anthony D. Joseph. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing.* Rapport technique, Berkeley, CA, USA, 2001. (Not cited.)

Peer-to-Peer Branch-and-Bound Algorithms for Computational Grids.

Abstract:

In the field of Combinatorial Optimization, the resolution to optimality of large instances of optimization problems through the use of Branch-and-Bound algorithms require a huge amount of computational resources. Nowadays, such resources are available from computing grids, which are sets of computing nodes geographically distributed over multiple sites. These parallel environments introduces multiples challenges related to the scalability, the heterogeneity of resources and the fault tolerance. Most of the existing approaches for the Branch-and-Bound algorithm are based on the Master-Slave paradigm where a central entity shares work units among slave entities in charge of processing them. Such an architecture represents an obstacle to scalability. In this thesis, we propose to face the challenges of grid environments and overcome this limitation by proposing an innovative and fully distributed approach based on the Peer-to-Peer paradigm. This architecture is based on a unique type of entity, a peer which is in charge of exploring its own local work pool and broadcasts global information to the network. We provide mechanisms to deal with the main tasks of the Branch-and-Bound algorithm : the load balancing, the diffusion of the best solution and the detection of the termination. Along with extensive experiments conducted on the Flow-Shop Scheduling Problem using the Grid'5000 Experimental Grid, we propose a formal proof of the correctness of our approach.In addition to this, we tackle a central issue when designing a Peer-to-Peer application : the impact of the P2P network topology on the performance of our approach. This aspect is often ignored in most of existing works, where only a predefined organization is chosen for the peers. The obtained results showed that the approach allows to deploy computing networks at extreme scales, involving hundreds of thousands of computing cores. Our final contribution consists in a Fault-Tolerant approach to deal with the dynamicity of the network (the volatility of computational resources). Results indicate that it faces efficiently various real-case and failure-intensive situations.

Keywords: Peer-to-Peer Computing, P2P, Parallel Branch-and-Bound, Fault Tolerance, Grid Computing, Large Scale Networks, Topologies, Network Protocols, Flow-Shop Scheduling Problem, Grid'5000, Termination Detection, Combinatorial Optimization, Exact Methods.